Task/Subtask IR70E
CDRL Sequence 07001-001
28 February 1991

(2)

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS) PROGRAM

The Cleanroom Engineering Software Development Process

Contract No. F19628-88-D-0032

Task IR70E - Cleanroom Development Process

Prepared for:

**Electronic Systems Division
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000**

Prepared by:

**IBM Federal Sector Division
800 North Frederick Avenue
Gaithersburg, MD 20879**

422280

311 08

This document has been approved for public release.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
2/28/91

3. REPORT TYPE AND DATES COVERED
Initial

4. TITLE AND SUBTITLE
The Cleanroom Engineering Software Development Process

5. FUNDING NUMBERS
F19628-88-D-0032 / 0002

6. AUTHOR(S)
Richard H. Cobb, Ara Kouchakdjian, Harlan D. Mills
Software Engineering Technology, Inc.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
IBM Federal Systems Company
800 North Frederick Avenue
Gaithersburg, MD 20879

8. PERFORMING ORGANIZATION REPORT NUMBER
07001-001

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Electronic Systems Center/AVK
Air Force Systems Command, USAF
Hanscom Air Force Base, MA 01731-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
N/A

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Cleared for public release

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Cleanroom provides software developers with the basis for developing software under statistical quality control. Software is functionally verified and certified, not tested, using sampling techniques, thus permitting software developers to assert a mean time to failure (MTTF) for the software modules they develop.

Under the STARS Contract, the IBM STARS team developed a process manual to assist software development organizations in adopting and installing the Cleanroom Engineering Software Development Process. This manual describes the process and verification activities required for performing a Cleanroom Engineering effort from the standpoint of specifiers, developers, certifiers, and managers.

The manual was developed to support process improvement programs directed at improving software quality and development productivity by incorporating some or all of the Cleanroom Engineering software development technologies into their current software development process and practices.

14. SUBJECT TERMS
Process, Process Management, Defined Process, Cleanroom, Software Engineering

15. NUMBER OF PAGES
302

16. PRICE CODE
N/A

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

Table of Contents

Figure List

Page

Section 5. State Data For Cleanroom Engineering

Section 8. Cleanroom Specification Team Practices

Section 9. Cleanroom Development Team Practices

Page

Section 10. Cleanroom Certification Team Practices

Section 1
A Cleanroom Engineering Process Model

This report has been prepared by Software Engineering Technology (SET). The objective of this report is to provide a manual that will help organizations utilize the Cleanroom technologies. SET received partial funding to complete this manual from the U.S. Defense Department under the DARPA STARS program (IBM P.O. 308334-PJ, Task IR-70 Extended). The task objective was:

> To develop a manual that software development managers can use to support process improvement programs directed at improving software quality and development productivity by incorporating some or all of the Cleanroom Engineering software development technologies into their current software development process and practices.

The manual is designed to help answer the following question that is typical of the reaction of many software development managers when they learn about Cleanroom,

> "The Cleanroom ideas sound fine, but how do we fit them into our current software development practices."

This section is divided into three parts. The first part is a brief reader's guide designed to introduce the reader to the manual. The second part is a brief summary of the principal findings resulting from this effort. The final part is a brief profile of the IR-70 Extended task.

1.1 Readers Guide

The manual has 9 sections and an Appendix A in addition to this introductory section. Section 2 - Section 6 define the Cleanroom Engineering process in a Box Structures framework.

Section 2 The Cleanroom Engineering: The Management Perspective

> This section reviews the Cleanroom technologies to establish a common level of understanding of the Cleanroom ideas for developing software under statistical quality control. This section concludes with a complete list of Cleanroom references.

Section 3 A Box Structures Model For Cleanroom Engineering

> A prerequisite to developing a useful process manual is the use of a model (abstraction) of the process. Such a model permits people who want to develop a specific realization of the process for some intended purpose to reason about the process abstractly. In this way a user can easily adopt the process to meet the individual concerns of the project by using the model. In this section a model of the full software development process incorporating the Cleanroom technologies is developed. Box Structures, which is one of the Cleanroom technologies, is used to develop and document the model.

Section 4 Process Basis For Cleanroom Engineering

This section fills in all the details of 25 interrelated processes defined for the Cleanroom process model. Using this section, organizations can develop a realization of the Cleanroom Engineering process for a specific project When developing the realization, the manager can specialize or adjust the model (1) to reflect organizational specific concerns, (2) to select only a subset of the Cleanroom Engineering practices or (3) to improve the process in a continuous program of process improvement.

Section 5 State Data For Cleanroom Engineering

A unique feature of this model is the concept of modeling the work-in-process as state data instead of finished products. This feature has the following advantages: (1) it makes the model easier to read and use, (2) it represents how developers actually work, and (3) it makes it easier to tailor the model to new and modified situations. This section summarizes the requirements for state data files to facilitate the development of state data files for a specific project.

Section 6 Responses For Cleanroom Engineering Projects

This is a short section which summarizes the thirteen responses that result from a software development project.

Previous sections have looked at the Cleanroom process in its entirety. Sections 7 - 10 examine at the Cleanroom process from the vantage point of people who use the Cleanroom practices - the engineering manager and three engineering teams.

Section 7 Managing A Cleanroom Project

This section looks at the model from the vantage point of a manager. The manager uses the model to organize a project by putting in place a concrete realization of the process and then using the realization to produce the desired software.

Section 8 Cleanroom Specification Team Practices

This section examines the process model from the vantage point of the Specification team. The Specification team is responsible for developing the specifications and verifying that if software is developed in a pipeline of accumulating increments, in accordance with the specifications, the software will achieve the mission assigned to the software.

Section 9 Cleanroom Development Team Practices

This section examines the process model from the vantage point of the Development team. The Development team is responsible for developing and verifying that the software as designed is in accordance with the specifications.

Section 10 Cleanroom Certification Team Practices

This section examines the process model from the vantage point of the Certification team. The Certification team is responsible for certifying the reliability of the software. Software reliability is defined by how often the software produces the output defined by the specifications or in terms of how long a user can expect to use the software before encountering a result that is not in accordance with the specifications.

Besides meeting the needs of the manager who wants to use Cleanroom Engineering on a software project, this manual provides assistance to two other audiences. First, managers and others who are trying to understand Cleanroom processes can learn about Cleanroom from a new perspective that some may find helpful. Second, process researchers who are interested in studying software development processes will find the manual helpful in two ways: (a) they can learn about Cleanroom, and (b) they can study how Box Structures were used to support the modeling effort so they can use these ideas in their own modeling efforts.

In the following paragraphs a list of the information which has not been published before is provided so the Reader familiar with Cleanroom can easily find the new information.

Section 2 Cleanroom Engineering: The Management Perspective

All the material in this section has been published before; but this is the first time all this material has been collected and published in a single, comprehensive description of Cleanroom Engineering.

Section 3 A Box Structures Model For Cleanroom Engineering
Section 4 Process Basis For Cleanroom Engineering

This material is all new since no process model of Cleanroom Engineering has ever been developed. Box Structures have never been used before to define a model of a process that is complex as software development. This modeling process has proven to be very successful so in addition to providing a useful model, these two sections provide an interesting case study.

Section 5 State Data For Cleanroom Engineering

The idea of treating the in-process work associated with software development as state data instead of products is a new idea. This treatment seems to be more representative of how people actually work and, as a result, permits the model to be considerably simpler.

Section 6 Responses For Cleanroom Engineering Projects

Same comments as for Sections 3 and 4.

Section 7 Managing A Cleanroom Project

All material in this section has been implied before in writings directed at the engineer, but they have never been stated from the perspective of the project manager.

Section 8 Cleanroom Specification Team Practices
Section 9 Cleanroom Development Team Practices
Section 10 Cleanroom Certification Team Practices

Much of this material has never previously been discussed in this form except in SET's Cleanroom education material. This is especially true for Sections 8 and 10.

1.2 Principal Findings

For the convenience of the reader, the principal findings of the effort that led to the preparation of the process manual are summarized in this section.

The Box Structures modeling process performed very well.

Evidence: Software development is a complex activity, and the model seems to be useful.

Recommendation: Box Structures should be used to support many more modeling efforts.

The modeling of in-process work as state data was both convenient and seems to represent the way people actually develop software.

Evidence: The simplicity of the model.

Recommendation: Organizations should begin to manage in-process work as state data. This will require some real-world experience to learn exactly how to make this transformation in thinking.

The process model appears to meet the needs of the manager who wants to incorporate Cleanroom into a project.

Evidence: Positive comments from people who have been exposed to the model.

Recommendation: The model should now be tested and perfected on real-world projects.

Cleanroom technologies simplify both the process of developing software and the management of software development activities.

Evidence: The model.

Recommendation: Cleanroom Engineering should be adopted by all organizations that want to improve the quality of their software and the productivity with which they develop software.

Cleanroom Engineering can be gracefully adapted by software engineering organizations.

Evidence: The model.

Recommendation: Organizations should use this manual to tailor a software development process that includes Cleanroom for their organization.

1.3 IR-70 Project Profile

SET, under a subcontract to IBM Federal Sector Division, initiated the adaptation of Cleanroom Software Engineering ideas and concepts to the STARS environment under Task IR-70. The first phase of the effort recast the Cleanroom ideas into the STARS environment (September 30, 1989, CDRL 1710 entitled STARS - Cleanroom Reliability: Cleanroom Ideas In The STARS Environment). The second phase was a case study involving development and certification of a software system consisting of a mixture of reused and newly written Ada modules (May 30, 1990, CDRL 1940 entitled A Case Study Using Cleanroom With Box Structures ADL). The third phase of the Task has involved the development of a Cleanroom Engineering Process manual.

The Cleanroom Engineering process for software development is based on three major activities: specification, development, and certification. Initially, the process manual was conceived as a document with separate sections (i.e., binders assigned CDRL's) to cover these activities. As the Cleanroom software development process was refined to provide a "what-to-do" road map for planning and controlling a Cleanroom software project, the separate binder approach was determined to be inadequate. The model that finally evolved as the best approach to present the process requires the definition of all the engineering processes used, the control structure between processes, the state data required to maintain the current state of design and processes used for maintaining the state data. This final IR-70 report presents an integration of the separate binders into a unified Cleanroom Engineering process described in a hierarchy of 25 subprocesses in which the specification, development, and certification activities play integrated roles.

1.4 Acknowledgments

SET would like to acknowledge the financial support received from the U.S. Defense Department under the DARPA STARS program and the encouragement received from the IBM STARS team in the preparation of this manual. SET also wants to acknowledge all the Cleanroom engineers who, through their questions and shared experiences, contributed to this manual by increasing our understanding of how to develop and certify software using the Cleanroom technologies.

Section 2
The Cleanroom Engineering Process: The Management Basis

2.1 Why Cleanroom Engineering?

2.1.1 Cleanroom Engineering Goals

The Cleanroom Engineering goal is to create an entirely new human capability for developing zero-defect software products and systems with high productivity.

The ultimate goal of the Cleanroom Engineer is to develop failure-free software. Cleanroom Engineers strive for zero-defect software by utilizing rigorous practices during design and development to implement a quality system and statistical quality control methods to certify reliability. These practices not only improve product quality but also increase development productivity.

Cleanroom Engineering of Software has been demonstrated as feasible and beneficial in both productivity and quality in software development. The idea driving the development of Cleanroom Engineering was that by first developing and then applying proper mathematical rigor to the engineering of software, essentially failure-free software could be developed. The effort started some 25 years ago at IBM. The Cleanroom ideas have been continuously refined, both as a result of theoretical investigations and field use. Section 2.4 contains references to 25 books and papers covering 20 years, many of which document this work. Progress has been significant. Today, several leading-edge organizations are using Cleanroom technologies to develop software.

Cleanroom Engineers are developing significant software systems of very high quality. There are several examples of widely used failure-free software (i.e., no user has yet encountered a failure). Current quality metrics in terms of latent defect rate per 1000 Lines Of Code (KLOC) at delivery for software development organizations are as follows: traditional practices (bottom-up design, structured code, defect removal through testing) 15-18 defects/KLOC; modern practices (top-down design, structured programming, design/code inspections, etc.) 2-4 defects/KLOC; and Cleanroom Engineering (0-1 defects/KLOC).

Users of Cleanroom are now targeting to release 100 KLOC systems that, more often than not, have no failures found during software use. They expect, as they move up the learning curve, to develop even larger failure-free systems. The surprise to many is that Cleanroom developers also increase their productivity along with achieving these much higher quality levels. Productivity rates of 750 lines of code per staff month are achieved with Cleanroom projects compared to the typical 75-250 lines of code per staff month being achieved by developers using modern programming practices. The reason? It is less costly to design the software in a manner that avoids failures than it is to test in quality into a failure-laden product (where defects have been designed in as a result of using inadequate engineering practices).

2.1.2 The Basis For Cleanroom Engineering

Cleanroom Engineering is made up of three major activities; namely, **Software Specification**, **Software Development**, and **Software Certification**. Software Specification and Software Development are based on well known formal methods in computer science and software engineering. Software Certification is based on statistical methods typically found outside computer science and software engineering, yet well known and practiced in other engineering and manufacturing fields.

Many people presently believe that software has nothing to do with Statistical Quality Control because software is deterministic, not statistical, with results that are either right or wrong. In this view, software is bound to have periodic failures; and "good" software just exhibits fewer failures than poor software. Statistical Quality Control in manufacturing or services assumes a correct design and deals with efforts to produce products consistent with such a design. In that case, the statistics deal with departures of the product from the design. With software the situation is reversed. In software the manufacture is practically perfect in terms of program compiling, linkage editing, etc. It is the design itself which is under question and the statistics deal with departures of the design from the specifications that are discovered during use.

In order to certify the reliability or quality of software, it is necessary to use a testing program that yields meaningful measurements which can serve as a basis for estimating how well the software as designed and implemented will satisfy its requirements in actual use. In the past few years significant progress has been made in recognizing how to use statistics to measure software quality.

The fields of statistics and software engineering are merging. As a result the software development industry is at the beginning of a very exciting time. This symbiosis is likely revolutionize software development, just as the symbiosis of statistics and manufacturing has revolutionized how factories are operated and managed. Cleanroom Engineers are now beginning this revolution.

Cleanroom Engineering develops software of **certified reliability** under **statistical quality control** in a **pipeline of increments** that accumulate into the specified software product. In the Cleanroom process, no program debugging is permitted before independent statistical usage testing of the increments as they accumulate into the final product. The Cleanroom development process provides rigorous methods of software specification, development and certification, through which disciplined software engineering teams are capable of producing low- or zero-defect software of arbitrary size and complexity. Such engineering discipline is not only capable of producing highly reliable software, but also the certification of the reliability of the software as specified.

Of course, humans are fallible, even in using sound mathematical processes such as functional verification, so software failures are possible during the Cleanroom certification process. However, there is a surprising power and synergy between functional verification and statistical usage testing. First, it has been found that functional verification scales up to large problems with high productivity and still leaves fewer errors than heuristic programming often leaves after unit and system testing combined. And second, it turns out that the mathematical errors left are much easier to find and fix during certification testing than errors left behind in debugging. This contrast has been measured to be a factor of five in practice [14].

2.1.3 Software Testing In This First Human Generation

2.1.3.1 Unit Testing As A Private Activity

In this first generation of software development and maintenance, the primary methods of software specification and design have been heuristic, going from informal natural language to formal programming language by trial and error. Unit testing is used without question and is regarded as a private activity for getting defects out of the small parts of programs before assembling and integrating them into larger parts. Subsequently, more defects are discovered in the larger parts as smaller parts are tied together. Finally, entire systems are frequently (almost always) delivered with more defects yet to be found. Users frequently find many more defects than the developers believed remained. As a result, many organizations have learned to expose new software products to a select few users for initial shakedown before widely distributing the products.

This first generation of heuristic methods and experiences seem to work. After all, products are built. Advanced software teams do better than others in using the best technical and management methods that can be found. And yet, even the advanced teams stub their software toes now and then. In fact, with all the programs written, more systems and products than casual observers might imagine are seriously delayed or abandoned, too error prone to be released. Hundreds of staff years may be involved, but the software still cannot be made to work correctly. Recently, an entire airline reservation system could not be made to work and was never put into operation. The first complete PL/1 compiler of a major computer company, involving more than a hundred staff years in development, was never released. Although the private unit debugging was done, the entire compiler could not be made to work. Several major PC upgrade products for word processing and financial analysis have been released more than a year behind their announcements, at a major cost to their producing companies.

The strange thing about most software disasters is that they were not seen as dangerous undertakings in their beginnings. Of course, any software effort is a bit risky because computer code is so detailed and programmers are a little unpredictable. "Nobody said it would be so hard at the outset," is a typical comment. New and better heuristics, especially supported by CASE tools, are coming available in object-oriented methods. With object-oriented methods, better approaches to high level designs and specifications seem possible. The software rubber meets the hardware road with the program code, and unit debugging is seldom questioned. At that stage, larger parts and entire products are tested, often with good records kept on the coverage testing, to ensure that every branch is exercised both ways and that all code is tested at least once. Yet in spite of the testing coverage, users often find unexpected levels of failures in operations, making the product marginal or unacceptable.

A barely recognized fact is that unit debugging is the most error prone activity in software development today. Developers fix user-observed failures by isolating and modifying what they believe is the cause for the failure, which is called a fault. Fixing a fault found is usually successful. But creating a new and deeper fault sometimes occurs as often as one time in five, and almost always as often as one time in ten. This occurs because debugging strives to ensure correct outputs, which lead to developers modifying code to produce a correct output, and not modifying the code to produce the correct rule that implements the function. As a result, large failure-filled software systems may never be debugged sufficiently to be released, even though extensive efforts are made.

2.1.3.2 A Historical Lesson in Typewriting

These experiences are not surprising in this first human generation of software development. They just seem part of the problem facing people in the field. Or are they? A hundred years ago people faced another set of problems in using the new typewriters (this practical invention occurred late in the 19th century): how to type text and tabular material without errors at reasonable rates.

Typewriters were special machines for special purposes. Executives hand wrote their own letters by and large, and assistants or secretaries did likewise. Typewriters were used to write reports and documents with relatively poor quality reprints compared to printing. They certainly did not replace assembling print for printing machines.

Typewriting was error prone. One had to look at the keys, of course, while typing, so a reasonable way was to memorize the text one sentence at a time. But in going back and forth between the text and the typewriter, small mistakes or lapses were very possible from time to time. Correcting a character, even a word, was acceptable; but correcting a missed sentence early on a typed page was better fixed by starting the page over.

With this background for almost a human generation of using typewriters, the new idea of **touch typing** (typing without looking at the keys) was a very strange one. "That's silly. Who could possibly do that?" In teaching typewriting, people who look at the keys can get useful work done in the very first day. In fact, they learn practically all there is to know about a typewriter in the very first day, and just need to get more practice and skill by typewriting. In teaching touch typing, people get no useful work done in the first day or even the first week. "Why would anyone spend time in such a useless activity?" Of course, touch typing turned out to be an internationally useful method that put typewriters into business offices on a mass basis. Typewriter makers improved their products in many ways, but the reason typewriters were made in such quantities was due to people knowing how to use them well rather than companies knowing how to make them well.

There is a lesson in touch typing for software development. In software, teaching a programming language and how to compile and execute programs allows people to write programs right away. Very likely, such programs will require considerable debugging, and many text books say just that. With more and more experience in programming alone or in teams, errors and unit debugging are just an accepted and integral part of traditional programming efforts.

People with the right education and training do not need to unit debug their software any more than people need to look at the keys when they type. Just as in teaching touch typing, much less trial-and-error programming is now done at the beginning in good software education. There is also much greater emphasis on formal methods in program specification, design and verification. When serious programming begins only after formal methods, very little debugging will be required because of more explicit design and verification from specifications. "Why not let the computers find the errors? Why make so much of a simple program?" For simple programs that may be a perfectly good question. But as programs become larger and more complex, computers do not find the errors, and much more time will be spent debugging than writing the code originally.

2.1.3.3 Zero Defect Software is Really Possible

In spite of the experiences of this first human generation in software development, **zero-defect** software is really possible. However, there is no foolproof logical way to know that software is zero defect. The proof is in using the product without ever finding any failures.

Mathematics is very helpful in creating software that executes with zero defects. It is insufficient to guarantee it, in part due to human fallibility in using mathematics, in part to mathematics itself in logical incompleteness. Statistics is also very helpful in creating software that executes zero defect, but it is also insufficient. For example, given a program that has been tested statistically without failures, (say for time

T of execution) a minimax argument permits a statement that estimates the Mean Time to Failure (MTTF) as 2T. The program may then be used many times T with no failures, and the MTTF goes up accordingly. As noted, the proof is in the usage.

Three illustrations of zero-defect software follow.

First, the U.S. 1980 Census was acquired by a nationwide network system of 20 miniprocessors. The system was controlled by a 25 KLOC program, which operated its entire ten months in field use with no failure observed. It was developed by **Paul Friday**, of the U.S. Census Bureau, using stepwise refinement and functional verification in Pascal. Mr. Friday was given the highest technical award of the U.S. Department of Commerce for that achievement.

Second, the IBM wheelwriter typewriter products released in 1984 are controlled by three microprocessors with a 65 KLOC program. It has had millions of users with no failures detected. The IBM team creating this software also used functional verification and extensive testing in a well managed environment to achieve this result.

Third, the U.S. space shuttle software of some 500 KLOC, while not completely zero defect, has been zero defect in all flights. The IBM team also used stepwise refinement and functional verification, and extensive testing to achieve that result. The space shuttle software is such a large, complex and visible product that there are real lessons in it. All programmers were required to complete a basic curriculum of six pass/fail courses in understanding programs as rules for mathematical functions, and functional verification of programs and modules [8].

Several illustrations of early Cleanroom projects follow.

The IBM COBOL Structuring Facility, a complex product of some 80K lines of PL/I source code, was developed in the Cleanroom discipline, with box structured design and functional verification but no debugging before usage testing and certification of its reliability.

The IBM COBOL Structuring Facility (SF) converts an unstructured COBOL program into a structured one of identical function. It uses considerable artificial intelligence to transform a flat structured program into one with a deeper hierarchy that is much easier to understand and modify. The product line was prototyped with Cleanroom discipline at the outset, then individual products were generated in Cleanroom extensions. In this development, several challenging schedules were defined for competitive reasons, but every schedule was met.

The COBOL/SF products have high function per line of code. The prototype was estimated at 100 KLOC by an experienced language processing group, but the Cleanroom developed prototype was 20 KLOC. The software was designed not only in structured programming, but also in structured data access. No arrays or pointers were used in the design; instead, sets, queues, and stacks were used as primitive data structures [15]. Such data structured programs are more reliably verified and inspected, and also more readily optimized with respect to size or performance, as required.

COBOL/SF, Version 2, consisted of 80 KLOC, 28 KLOC reused from previous products, 52 KLOC new or changed, designed and tested in a pipeline of five increments [7], the largest was over 19 KLOC. A total of 179 corrections were required during certification, under 3.5 corrections per KLOC for the new code with no developer execution. The productivity of the development was 740 LOC per staff month, including all specification, design, implementation, certification and management, in meeting a very short deadline.

A version of the USAF HH60 (helicopter) flight control program of over 30 KLOC was also developed using Cleanroom. The HH60 flight control program was developed on schedule. Programmers' morale went from quite low at the outset ("why us?") to very high on discovering their unexpected capability in accurate software design without debugging. Effort expended to fix errors went down by a factor of 5. The twelve programmers involved had all passed the pass/fail course work in mathematical (functional) verification of the IBM Software Engineering Institute, but were provided a week's review as a team for the project. The testers had much more to learn about certification by objective statistics [5].

The Coarse/Fine Attitude Determination Subsystems (CFADS) of the Upper Atmospheric Research Satellite (UARS) Attitude Ground Support System (AGSS) of some 30 KLOC has been developed in Fortran with partial Cleanroom at NASA. Sixty-two percent of the subroutines, which averaged 258 source lines each, compiled successfully the first time the testers tried to compile them, and all but one of the rest compiled successfully on the second attempt. Compared with well measured related systems in the same environment, the failure rate was down by a factor of 2 while the productivity was up by 70% [6].

V. R. Basili and F. T. Baker introduced Cleanroom ideas in an undergraduate software engineering course at the University of Maryland, assisted by R. W. Selby. As a result, a controlled experiment in a small software project was carried out over two academic years, using fifteen teams with both traditional and Cleanroom methods. The result, even on first exposure to Cleanroom, was positive in the production of reliable software, compared with traditional methods [24].

Cleanroom projects have been carried out at the University of Tennessee, under the leadership of J. H. Poore [18] and at the University of Florida under H. D. Mills. At Tennessee, a team has achieved the high level of performance expected of Cleanroom and additional students continue to be trained in the Cleanroom concepts. At Florida, seven teams of undergraduates produced uniformly successful systems for a common structured specification of three increments. It is a surprise for undergraduates to consider software development as a serious engineering activity using mathematical verification instead of debugging, since software development is typically introduced primarily as a trial-and-error activity with no real technical standards.

2.1.4 What is Cleanroom Engineering of Software?

The **Cleanroom Engineering** process develops software of **certified reliability** under **statistical quality control** in a **pipeline of increments**, with **box structured design** and **functional verification** but **no program debugging** permitted before **independent statistical usage testing** of the increments. It provides rigorous methods for software **specification, development** and **certification** that are capable of producing low- or zero-defect software of arbitrary size and complexity. Box structured design is based on a **Parnas** usage hierarchy of modules. Such modules, also known as data abstractions or objects, are described by a set of operations that may define and access internally stored data. Functional verification is based on the fact that any program or program part is a **rule for a mathematical function**. It may not be the function desired, but it is a function.

The term Cleanroom is taken from the hardware industry to mean an emphasis on preventing errors, rather than allowing errors to appear and removing them later (of course any errors discovered should be removed). Cleanroom Engineering involves rigorous methods that enable greater control over both product and process. The Cleanroom process not only produces software of high reliability and high performance, but does so while yielding high productivity and schedule reliability. The intellectual control provided by the rigorous Cleanroom process allows both technical and management control.

Cleanroom Engineering achieves **statistical quality control** over software development by strictly **separating** the **development** process from the **testing** process in a pipeline of incremental software development. There are three major engineering activities in the Cleanroom Engineering process [3, 7, 14]:

First, a Specification team creates an incremental specification that defines a pipeline of software increments that accumulate into the final software product, which specification includes the statistics of its use as well as its function and performance requirements;

Second, a Development team designs and codes increments specified using box-structured design and stepwise refinement with functional verification of each increment, delivers them for certification with no debugging beforehand, and provides subsequent correction for any failures that may be uncovered during certification;

Third, a Certification team uses statistical testing and analysis for the certification of the software reliability to the usage specification, notification to developers of any failures discovered during certification, and subsequent revalidation as failures are corrected.

As noted, there is an explicit feedback process between certification and development on any failures found in statistical usage testing. This feedback process provides an objective measure of the reliability of the software as it matures in the development pipeline. It does, indeed, provide a statistical quality control process for software development not available in this first human generation of trial and error programming.

2.1.5 Cleanroom Software Engineering Methods

Cleanroom Engineerinɡ proᵛides a set of rigorous methods for **software development under statistical quality control**, based on sound mathematical and statistical principles. While millions of people are involved in software, most of them regard software development as an intuitive, heuristic activity. They do not imagine software engineering as a mathematically based subject with complete rigor being possible. But software engineering should be and can be a mathematics based activity. When mathematical rigor is applied both quality and productivity increase. Nor can they imagine software engineering based on statistics since computers are completely deterministic in behavior. And yet the usage of software is statistical in nature.

Software is a human generation old, while mathematics is many human generations old. Although not understood early or widely, software has direct mathematical foundations because of the very deterministic behavior of computers. A computer program is a rule for a mathematical function, mapping all possible initial states into final states. Such functions are very complex compared to functions in physical science and engineering, and traditional mathematical notation is very insufficient. But sufficient mathematical notation is emerging in computer science and software engineering for dealing with the syntax and semantics of programs and their functions.

As an example of deep and useful mathematics, place notation and long division moved arithmetic from intuition and heuristics to rigorous methods a thousand years ago in the western world. As a result, school children today can outperform Archimedes and Euclid in arithmetic. Similar movement is possible in software today.

Statistics is another subject of longer professional development than software. But only a hundred years ago, statistics was intuitive and heuristic, even though rigorous arithmetic was used in creating sums and averages. Yet in this time, statistics has become a rigorous, mathematics based subject, often finding counterintuitive results using statistics in specific topics. The application of statistics makes it possible for software developers to predict with confidence the quality level of the software when it is fully developed quite early in the development life cycle.

Cleanroom Engineering not only puts software development under statistical quality control, but takes debugging out of the list of developer activities, instead using mathematical reasoning before independent testing and certification. Just as typists looked at the keys when typewriters first came out, programmers have felt the need to debug programs in this first human generation of programming. But while counterintuitive at the time, typists went to touch typing with both higher productivity and fewer errors. In the same way, well educated software engineers can create software with no execution or debugging before it is tested by independent test and certification engineers, with the product having higher productivity and much greater quality.

2.1.6 Dealing with Human Fallibility

Humans are fallible, even in using sound mathematical processes in functional verification. Finding software failures is possible during the certification process. There is a surprising **power and synergism** between **functional verification** and **statistical usage testing** [14]. First, functional verification can be scaled up for high productivity and still leave no more errors than heuristic programming often leaves after unit and system testing combined. Second, it turns out that the mathematical errors left are much easier to find and fix during testing than errors left behind in debugging, measured at a factor of five in practice [14]. Mathematical errors usually turn out to be simple oversights in the software, whereas errors left behind or introduced in debugging are usually deeper in logic or wider in system scope than those fixed. As a result, statistical usage testing not only provides a formal, objective basis for the certification of reliability under use, but also uncovers the errors of mathematical fallibility with remarkable efficiency.

A major discovery, in Cleanroom Engineering is the ability of well prepared and motivated people to create nearly defect free software before any execution or debugging, with less than five defects per thousand lines of code. Such code is ready for usage testing and certification with no unit debugging by the designers. In this first human generation of software development it has been counterintuitive to expect software with so few defects at the outset. Traditional heuristic programming practices leave 50 - 60 defects per thousand lines of code, then the defects are reduced to 15 -18 by testing. Software developers who use more modern heuristic practices leave only 20 - 40 defects per thousand lines of code which are reduced to 2 - 4 by testing.

The mathematical foundations for Cleanroom Engineering come from the deterministic nature of computers themselves. As noted, a computer program is no more and no less than a rule for a mathematical function [8, 9]. Such a function need not be numerical, of course, and most programs do not define numerical functions. But for every legal input, a program directs the computer to produce an output, whether correct as specified or not. And the set of all such input, output pairs is a mathematical function. A more intuitive way to view a program in this first generation is as a set of instructions for specific executions with specific input data. While correct, this view misses a point of reusing well known and tested mathematical ideas, regarding computer programming as new and private art rather than more mature and public engineering.

With these mathematical foundations, software development becomes a process of constructing rules for functions that meet required specifications, which need not be a trial and error programming process. The functional semantics of a structured programming language can be expressed in an algebra of functions with function operations corresponding to program sequence, alternation, and iteration [8]. The systematic top down development of programs is mirrored in describing function rules in terms of algebraic operations among simpler functions, and their rules in terms of still simpler functions until the rules of the programming language are reached. It is a new mental base for most programmers to consider the complete functions needed, top down, rather than computer executions for specific data.

2.1.7 The Power of Usage Testing over Coverage Testing

The insights and data of Adams [1] in the analysis of software testing, and the differences between software errors and failures, give entirely new understandings in software testing. Since Adams has discovered an amazingly wide spectrum in failure rates for software errors, it is no longer sensible to treat errors as homogeneous objects to find and fix. Finding and fixing errors with high failure rates produces much more reliable software than finding and fixing random errors, which generally have average or low failure rates.

The major surprise in Adams' data is the relative power of finding and fixing errors in usage testing over coverage testing, a factor of 30 in increasing MTTF. That factor of 30 seems incredible until the facts are worked out from Adams' data. But it explains many anecdotes about experiences in testing. In one such experience, an operating systems development group used coverage testing systematically in a major revision and for weeks found mean time to abends in seconds. It reluctantly allowed user tapes in one weekend, but on fixing those errors, found the mean time to abends jumped literally from seconds to minutes.

The Adams data is given in Table 2.1.7.1 from [1]. It describes distributions of failure rates for errors in nine major IBM products, including the major operating systems, language compilers, data base systems. The uniformity of the failure rate distributions among these very different products is truly amazing. But even more amazing is a spread in failure rates over three orders of magnitude, from 19 months to 5000 years (60 K months) calendar time in MTTF, with about a third of the errors having a MTTF of 5000 years, and 1% having a MTTF of 19 months.

Table 2.1.7.1 Distributions of Errors (In %) Among Mean Time to Failure (MTTF) Classes

	MTTF in K months							
	60	19	6	1.9	.6	.19	.06	.019
Product								
1	34.2	28.8	17.8	10.3	5.0	2.1	1.2	.7
2	34.2	28.0	18.2	9.7	4.5	3.2	1.5	.7
3	33.7	28.5	18.0	8.7	6.5	2.8	1.4	.4
4	34.2	28.5	18.7	11.9	4.4	2.0	.3	.1
5	34.2	28.5	18.4	9.4	4.4	2.9	1.4	.7
6	32.0	28.2	20.1	11.5	5.0	2.1	.8	.3
7	34.0	28.5	18.5	9.9	4.5	2.7	1.4	.6
8	31.9	27.1	18.4	11.1	6.5	2.7	1.4	1.1
9	31.2	27.6	20.4	12.8	5.6	1.9	.5	.0

With such a range in failure rates, it is easy to see that coverage testing will find the very low failure rate errors a third of the time with practically no effect on the MTTF by the fix, whereas usage testing will find many more of the high failure rate errors with much greater effect. Table 2.1.7.2 shows the relative effectiveness of making fixes found by usage testing compared to coverage testing, in terms of increased MTTF. Table 2.1.7.2 develops the change in failure rates for each MTTF class of Table 2.1.7.1, because it is the failure rates of the MTTF classes that add up to the failure rate of the product.

Table 2.1.7.2 Error Densities and Failure Densities in the MTTF Classes of Table 2.1.7.1

Property								
M	60	19	6	1.9	.6	.19	.06	.019
ED	33.2	28.2	18.7	10.6	5.2	2.5	1.1	.5
ED/M	.6	1.5	3.1	5.6	8.7	13.2	18.3	26.3
FD	.8	2.0	3.9	7.3	11.1	17.1	23.6	34.2
FD/M	0	0	1	4	18	90	393	1800

First, in Table 2.1.7.2, line 1, denoted M (MTTF), is repeated directly from Table 2.1.7.1, namely the mean time between failures of the MTTF class. Next, line 2, denoted ED (Error Density), is the average of the error densities of the 9 products of Table 2.1.7.1, column by column, which represents a typical software product. Line 3, denoted ED/M, is the contribution of each class, on the average, in reducing the failure rate by fixing the next error found by coverage testing (1/M is the failure rate of the class, ED the probability a member of this class will be found next in coverage testing, so their product, ED/M, is the expected reduction in the total failure rate from that class). Now ED/M is also proportional to the usage failure rate in each class, since failures of that rate will be distributed by just that amount. Therefore, this line 3 is normalized to add to 100% in line 4, denoted FD (Failure Density). It is interesting to note that Error Density (ED) and Failure Density (FD) are almost reverse distributions, Error Density about a third at the high end of MTTF's and Failure Density about a third at the low end of MTTF's. Finally, line 5, denoted FD/M, is the contribution of each class, on the average, in reducing the failure rate by fixing the next error found by usage testing.

The sums of the two lines ED/M and FD/M turn out to be proportional to the decrease in failure rate from the respective fixes of errors found by coverage testing and usage testing, respectively. Their sums are 77.3 and 2306, with a ratio of about 30 between them. That is the basis for the statement of their relative worth in increasing MTTF. It seems incredible at first glance, but that is the number!

To see that in more detail, consider, first, the relative decreases in failure rate R in the two cases:

Fix next error from coverage testing

$$R \rightarrow R - (\text{sum of ED/M values})/(\text{errors remaining}) \\ = R - 77.3/E$$

Fix next error from usage testing

$$R \rightarrow R - (\text{sum of FD/M values})/(\text{errors remaining}) \\ = R - 2306/E$$

Next, the increase in MTTF in each case will be

$$1/(R - 77.3/E) - 1/R = 77.3/(R*(E*R - 77.3))$$

and

$$1/(R - 2306/E) - 1/R = 2306/(R*(E*R - 2306))$$

In these expressions, the numerator values 77.3 and 2306 dominate, and the denominators are nearly equal when E*R is much larger than 77.3 or 2306 (either 77.3/(E*R) or 2306/(E*R) is the fraction of R reduced by the next fix and is supposed to be small in this analysis). As noted above, the ratio of these numerators is about 30 to 1, in favor of the fix with usage testing.

2.2 Mathematical Basis of Software Design

2.2.1 Box Structured Software System Design

Box structured design is based on a **Parnas** usage hierarchy of modules [21, 22]. Such modules, also known as data abstractions or objects, are described by a set of operations that may define and access internally stored data. In Ada, such modules are defined as **packages**, with operations defined by the calls of the procedures and functions of the packages, and internal data declared in the package.

Stacks, queues, and sequential or random access files provide simple examples of such modules or packages. Part of their discipline is that internally stored data cannot be accessed or altered in any way except through the explicit operations of the package. It is critical in box structured design to recognize that packages exist at every level from complete systems to individual program variables. It is also critical to recognize that a verifiable design must deal with a usage hierarchy rather than a parts hierarchy in its structure. A program that stores no data between invocations can be described in terms of a parts hierarchy of its smaller and smaller parts, because any use depends only on data supplied it on its call with no dependence on previous calls. But a specific realization of a package, say a queue, will depend not only on the present call and data supplied it, but also on previous calls and data supplied then.

The parts hierarchy of a structured program identifies every sequence, alternation, and iteration (say every begin-end, if-then-else, while-loop) at every level. It turns out that the usage hierarchy of a system of packages (say an object oriented design with all objects identified) also identifies every call (use) of every operation of every package. The semantics of the structured program are defined by a mathematical function for each sequence, alternation, and iteration in the parts hierarchy. That doesn't quite work for the operations of packages because of usage history dependencies. But there is a simple extension for packages that does work. It is to model the behavior of a package as a **state machine**, with its calls of its several operations as inputs to the common state machine. Then the semantics of such a package is defined by the **transition function** of its state machine (with an initial state). When the operations are defined by structured programs, the semantics of packages becomes a simple extension of the semantics of structured programs.

While theoretically straightforward, the practical design of systems of Parnas modules [22, 23] (object oriented systems) in usage hierarchies can seem quite complex on first exposure. It seems much simpler to outline such designs in parts hierarchies and structures, for example in data flow diagrams, without distinguishing between separate usages of the same module. While that may seem simpler at the moment, such design outlines are incomplete and often lead to faulty completions at the detailed programming levels. In spite of their common use in this first human generation of system design, data flow diagrams should only be used within rigorous design methods rather than leaving critical requirements to details with incomplete specifications.

In order to create and control such designs based on usage hierarchies in more practical ways, their box structures provide standard, finer grained subdescriptions for any package of three forms, namely as **black boxes**, as **state boxes**, and as **clear boxes**, defined as follows [12, 16, 17].

Black Box: External view of a function, describing its behavior as a mathematical function from historical sequences of stimuli to its next response.

State Box: Intermediate view of a function, describing its behavior by use of an internal state and internal black box with a mathematical function from historical sequences of stimuli and states to its next response and state, and an initial internal state.

Clear Box: Internal view of a function, describing the internal black box of its state box in a usage control structure of other Parnas packages; such a control structure may define sequential, alternative, iterative or concurrent use of the other packages.

Box structures enforce completeness and precision in design of software systems as usage hierarchies of Parnas packages. Such completeness and precision lead to pleasant surprises in human capabilities in software engineering and development. The surprises are in capabilities to move from system specifications to design in programs without the need for unit/package testing and debugging before delivery to system usage testing. The division of concerns into three views of the system allows the developers to keep intellectual control over the subsystem to be developed. In this first generation of software development, it has been widely assumed that trial and error programming, unit testing and debugging were necessary. But well educated, well motivated software professionals are, indeed, capable of developing software systems of arbitrary size and complexity without program debugging before system usage testing [7].

2.2.2 Stepwise Refinement and Functional Verification

Once the design is complete, the clear box at each level is expanded to code to fully implement the defined function rule for the black box function at that level by stepwise refinement, as introduced by Wirth [25]. Following each expansion, functional verification is used to help structure a proof that the expansion correctly implements the specification. The nature of the proof revolves around the fact that a program is a rule for a function and the specification for the program is a relation or function. What must be shown in the proof is that the rule (the program) correctly implements the function (the specification) for the full range of the specification and no more. Linger, Mills and Witt [8] have developed a correctness theorem which defines what must be shown to prove that a program is equivalent to its specification for each of the structured programming language constructs. The proof strategy is subdivided into small parts which easily accumulate into a proof for a large program. Experience indicates that people are able to master these ideas and construct proof arguments for very large software systems.

The Development team expands each clear box in the usage hierarchy into the selected target code using stepwise refinement and functional verification. As the Development team designs and implements the software, it is held collectively responsible for the quality of the software.

In describing the activities of software development, no mention is made of testing or even of compilation. The Cleanroom Development team does not test or even compile. They use mathematical proofs (functional verification) to demonstrate the correctness of programming units. Testing and measuring failures by program execution is the responsibility of the Certification team.

2.2.3 The Mathematical Basis for Functional Verification

Any program or program part is a **rule** for a **mathematical function**. It may not be the function desired, but it is a function. In structured programs, the rules are direct in form, building program rules out of just two function building operations: first, **function composition** which corresponds to sequential execution of program parts, and second, **disjoint function union** which corresponds to alternative execution of one program part or another, as in **if** or **case** structures. Program iteration uses no more than these two operations together, and function recursion provides a useful view of an iteration process.

As noted, any program part or total program defines a single, possibly complex function. The function is seldom a numerical function in classical terms. Even numerical programs must deal with finite sets of numbers in which overflow and roundoffs depart from classical number systems. Given the text or name of a program or program part in whatever language, say a program in Ada

```
Alpha =     procedure Beta
            is
            ...
            begin
            ...
            end Beta;
```

the **program function** will be denoted by brackets [ ] around the name or text, as

```
[Alpha] =   [procedure Beta
            is
            ...
            begin
            ...
            end Beta;]
```

In this case [Alpha] is a set of ordered pairs.

$$[\text{Alpha}] = \{\langle X, Y\rangle \mid \text{Given initial state } X\text{, Alpha will produce final state } Y\}$$

The function [Alpha] is determined by Ada text, but is independent of the language Ada. The same function can be defined in Fortran text, COBOL text, etc.

2.2.4 Functional Verification of Program Parts

From programs to program parts, starting with simple assignment statements, such as

```
x := y;
```

in Ada, the **program part function**

```
[x := y;]
```

takes its initial data state to its final data state. If legal, it will change the value of x in the final state to the value of y in the initial state and change no other values of variables in the initial state. If illegal, the final state may be quite different than the initial state, possibly with both x and y disappearing, as well as other variables, in terminating the entire program execution. So assignment statements have simple function parts when legal, but possible more complex function parts when illegal. In summary, the function [x := y;] is a set of ordered pairs with second members determined uniquely by the first members

$$\begin{aligned}[x := y;] = {} & \{\langle\langle x, y, \ldots\rangle, \langle y, y, \ldots\rangle\rangle \mid x := y; \text{ is legal}\} \\ & \cup \{\langle\langle x, y, \ldots\rangle, \langle ???\rangle\rangle \mid x := y; \text{ is illegal}\}\end{aligned}$$

where ??? will be determined by other aspects of the initial state. Illegal situations will be suppressed in what follows for sake of brevity. In more direct function notation, dealing only with the legal situation,

$$[x := y;](\langle x, y, \ldots\rangle) = \langle y, y, \ldots\rangle$$

in which the function **argument** $\langle x, y, \ldots\rangle$ produces the function **value** $\langle y, y, \ldots\rangle$.

Next, for a sequence of statements, such as

x := y; y := z; z := x;

in Ada, the part function

$$[x := y;\ y := z;\ z := x;]$$

will alter values of x, y, z as a composition of the three individual assignment functions

$$[x := y;]\circ[y := z;]\circ[z := x;].$$

That is, beginning with an initial state as argument, the first assignment function gives a new state as value

$$[x := y;](\langle x, y, z, \ldots\rangle) = \langle y, y, z, \ldots\rangle,$$

the second assignment function uses this value as an argument

$$[y := z;](\langle y, y, z, \ldots\rangle) = \langle y, z, z, \ldots\rangle,$$

and the third assignment function uses this last value as argument

$$[z := x;](\langle y, z, z, \ldots\rangle) = \langle y, z, y, \ldots\rangle.$$

That is, the composition function is a nested set of simpler functions that evaluate as

$$\begin{aligned}
&([x := y;]\circ[y := z;]\circ[z := x;])(\langle x, y, z, \ldots\rangle)\\
&= [z := x;]([y := z;]([x := y;](\langle x, y, z, \ldots\rangle)))\\
&= [z := x;]([y := z;](\langle y, y, z, \ldots\rangle))\\
&= [z := x;](\langle y, z, z, \ldots\rangle)\\
&= \langle y, z, y, \ldots\rangle
\end{aligned}$$

as worked out just above. In summary, this composition function will interchange the values of y and z and leave x with the initial value of y, not changing any other data in the initial state.

Finally, for an alternation statement, such as

if x > y then y := z else x := z end if;

in Ada, the part function will execute either the then part or else part, so that

$$
\begin{aligned}
&[\text{if } x > y \text{ then } y := z; \text{ else } x := z; \text{ end if;}]\\
&= (x > y \rightarrow [y := z;] \mid x <= y \rightarrow [x := z;])\\
&= [y := z; \mid x > y] \cup [x := z; \mid x <= y]
\end{aligned}
$$

where the expression $[y := z; \mid x > y]$ means the function $[y := z;]$ with its domain restricted to the condition $x > y$. That is, the part function is a union of disjoint functions.

2.3 Software Engineering Under Statistical Quality Control

The statistical foundations for Cleanroom Engineering come from adding usage statistics to software specifications, along with function and performance requirements [3, 5, 14]. Such usage statistics provide a basis for measuring the reliability of the software during its development, and thereby measuring the quality of the design in meeting functional and performance requirements. A more usual way to view development in this first generation is as a difficult-to-predict art form.

2.3.1 Statistical Certification

Cleanroom statistical certification of software involves, first, the **specification of usage statistics** in addition to function and performance specifications. Such usage statistics provide a basis for assessing the reliability of the software being tested under expected use. An efficient estimation process has been developed for projecting Mean Time to Failure (MTTF) of software under test while also under correction for previously discovered failures [5].

As each specified increment is completed by the designers, it is delivered to the certifiers, who combine it with preceding increments, for testing based on usage statistics. As noted, the Cleanroom construction plan must define a sequence of nested increments which are to be executed exclusively by user commands as they accumulate into the entire system required. Each subsequence represents a subsystem complete in itself, even though not all the user function may be provided in it. For each subsystem, a certified reliability is defined from the usage testing and failures discovered, if any.

It is characteristic that each increment goes through a maturation process during the testing, becoming more reliable from corrections required for failures found, serving thereby as a stable base as later increments are delivered and integrated to the developing system. For example, the HH60 flight control program had three increments [5] of over 10 KLOC each. Increment 1 code required 27 corrections for failures discovered in its first appearance in increment 1 testing, but then only 1 correction during increment 1/2 testing, and 2 corrections during increment 1/2/3 testing. Code in increment 2 required 20 corrections during its first appearance in increment 1/2 testing, and only 5 corrections during increment 1/2/3 testing. Increment 3 code required 21 corrections on its first appearance in increment 1/2/3 testing. In this case, 76 corrections were required in a system of over 30 KLOC, under 2.5 corrections per KLOC for verified and inspected code, with no previous execution or debugging.

In the certification process, it is not only important to observe failures in execution, but also the times between such failures in execution of usage representative statistically generated inputs. Such test data must be developed to represent the sequential usage of the software by users, which, of course, will account for previous outputs seen by the users and what needs the users will have in various circumstances. The state of mind of a user and the current need can be represented by a stochastic process determined by a state machine whose present state is defined by previous inputs/outputs and a statistical model that provides the next input based on that present state [14].

2.3.2 Certification Tasks

In parallel with the Cleanroom Development team, the Cleanroom Certification team prepares to certify the software up to and including the increment being developed by the Development team. The Certification team uses the usage profile and the portion of the specification that is applicable to the increments to be verified to prepare test cases including expected outputs to tests.

When the Development team has completed an increment, the Certification team creates one or more successive **versions** of the **accumulated system** up through this increment. For each version the Certification team compiles the increment, combines it with previous increments, and certifies the accumulated system through this version. If failures are encountered in the certification of a version, they are returned to the Development team for analysis and engineering changes to whatever increments are causing the failures. While failures are likely to be caused by the latest increment added, previous increments may be at fault and changed as well, as noted in HH60 experience. Each redelivery of changed increments defines a new version. If no failures are encountered in the certification of a version, no additional versions are required.

For each version of the accumulating system, tests are constructed at random in accordance with the specified usage statistics profile and then exercised. Test results are compared to the expected output and either a failure occurred or the result was correct. The execution time between successive failures is a sample of the MTTF for that version. All such samples of the MTTF for the version are used to estimate the MTTF for the future, corrected version of the software.

2.3.3 The Certification Model

The estimate for the MTTF of a version is based on a **certification model** which describes the MTTF as an algebraic expression based on the initial MTTF, say MTTF_0, at the start of certification, the number of independent failures that have been addressed by engineering changes, say EC, and the average MTTF rate of improvement per failure addressed, say ROI. The certification model is

$$\mathrm{MTTF} = \mathrm{MTTF}_0 \,(\, \mathrm{ROI}^{\mathrm{EC}} \,)$$

where MTTF_0 and ROI are to be estimated from the samples of MTTF and the known values of EC for each sample. The value of EC is known for each failure sample in a given version, namely the number of engineering changes preceding the construction of the version in which the failure occurred.

When common software development methods and skills are used across increments, the certification model can be used across versions with growing increments. With significant reuse of software modules and independent estimates of their MTTF, specific statistical analysis should be employed. What follows assumes common methods and skills in the software development. For certain systems, other measurements than MTTF may be desired. For example, the reliability of systems with independent start ups for specific missions may be better estimated on the basis of fraction of total missions that are failure free. Such statistics can be developed for what is needed.

This certification model has been used in real projects with statistical testing to obtain reasonable results. Of course there is no logical or exact way to verify any such model. But statistical reasoning provides a basis of confidence for using the certification model. There is a paradox in statistical testing. If software has a large number of failures, the statistics can become more precise; if software failures are rare, or never occur, the statistics are very imprecise. The paradox is that the best software has the least precise statistics. The response to this paradox is that the continued usage of the software is sampling itself and adds to the precision of the statistics. If software is entirely failure free, that fact cannot be established by statistical testing. Failure-free usage of the software becomes part of the statistics.

Since no failures detected gives no absolute proof that failures may never occur later, the MTTF must still be estimated as finite. In this case, the MTTF will be estimated at double the elapsed time with no failures detected. If failures do appear infrequently, the variability of times between failures will be very great; in fact the statistical standard deviation of times between failures will be the same as the mean time between failures. So estimates of MTTF are subject to considerable variability from the actual MTTF. At first this may sound as if the statistics are not very useful. In fact, the statistics are useful in entirely new ways, especially when compared with the lack of statistics. There are many counterintuitive facts emerging about statistical testing. For example, as shown previously, fixing a failure from random usage testing will improve the time to next failure 30 times as much as fixing a failure from random coverage testing.

2.3.4 The Certification Process

The certification of software is based on a statistical estimation process for the MTTF of a version based on the certification model above which describes the MTTF as an expression

$$\mathrm{MTTF} = \mathrm{MTTF}_0 (\mathrm{ROI}^{\mathrm{EC}})$$

where MTTF_0 and ROI are to be estimated from the samples of MTTF and the value of EC for each sample.

As developed in [6], three major alternatives have been considered for the estimation process, namely 1) a maximum likelihood estimator, 2) a least squares estimator, and 3) a logarithmic least squares estimator. The third alternative for estimation has turned out the most powerful, namely to redescribe the relation above in logarithmic form

$$\ln \mathrm{MTTF} = \ln \mathrm{MTTF}_0 + \mathrm{EC} (\ln \mathrm{ROI})$$

which is linear in the two variables $\ln \mathrm{MTTF}_0$ and ln ROI.

During the certification process for a version, the dynamics of change indicates how much more test time will be required to certify the software to a required MTTF. The term ROI is the factor by which addressing the next failure will increase the current MTTF. This provides an estimate on how many failures must be addressed to reach a required MTTF, and how much time will be required to find such failures. If, conceivably, ROI goes below one that means the new version of the system is worse than the previous version and serious management concern is needed. It is expected and desirable that the value of ROI is constant so MTTF increases monotonically. The value of ROI decreases when frequent failures are encountered late. Failures found early are not expensive in terms of eventually obtaining a high value for MTTF with a reasonable testing budget, but if ROI drops late in the certification process it can take a large number of additional tests to achieve a desired MTTF.

Once the MTTF for a version has been estimated a decision about the next action must be taken. The following actions are possible:

(1) address the observed failures and continue to certify the software,

(2) stop certification because the software has reached the desired reliability for this stage of testing, or

(3) stop certification and redesign the software because the failure rate is too high or the failures are too serious.

The process described above continues until all the increments are complete. When all the increments are complete and tested, the software can be deployed. Following deployment the operations and maintenance phases for the software are entered. These phases are not discussed further here except to observe that 1) operations provide actual testing for continued estimates of the MTTF to check against what has been certified during development, and 2) the maintenance phase will be much simpler for Cleanroom developed software than for heuristically developed software due to higher quality of the software and the design and development trail.

2.4 Human Issues of Cleanroom Engineering

In the previous sections many of the ideas that have led to the definition of the processes and practices that form the basis of Cleanroom Engineering have been introduced. In this section the integrated suite of Cleanroom Engineering processes and the associated rigorous engineering practices that facilitate the specification, development and certification of high-quality software are summarized.

2.4.1 Cleanroom Teams

As noted above, Cleanroom projects are performed by three teams based on a separation of concerns. In more detail the three teams are:

Specification Team

> The Specification team prepares, enhances and maintains the working specifications. The Specification team is responsible for verifying that if the software is implemented in accordance with the specification that the software will perform its mission in accordance with expectations. That means the software will provide complete user satisfaction in terms of providing the target return on investment and will satisfy the usability criteria established for direct users, while performing its intended mission.

Development Team

> The Development team designs and implements the software increment by increment in accordance with the specification and verifies that each increment satisfies the proportion of the specification assigned to that increment. The Development team uses: (a) Box Structures to design the increment and (b) Stepwise refinement and functional verification to implement and perform unit verification on the increment.

The Development team is responsible for the quality of the increment. There is no individual responsibility for development errors since the entire team participates in proof conversations so every team member has the opportunity and duty to detect and eliminate design errors.

Certification Team

The Certification team certifies that the accumulating increments satisfy the specification by performing independent usage testing with tests they have developed in accordance with the expected usage profile. The Certification team uses statistical quality control techniques to certify the software by

Preparing test scenarios and expected results
Determining the sampling plan
Performing configuration management on the accumulating software
Conducting tests on accumulating increments
Transferring failures to the Development team for resolution
Recording failure data
Estimating the reliability of each increment

These three engineering teams are responsible to an engineering manager who is responsible for leading software specification, development and certification efforts. In addition to leading, the engineering manager plans and controls the activities of the teams.

2.4.2 Developing The Specification

A Cleanroom software development project is organized as depicted in Figure 2.4.2.1. The project begins with the development of a specification and continues until the full software system has been certified as meeting its specifications. A Cleanroom Engineering project begins when a decision is made to develop some software and ends when a decision is made to accept the software for deployment. It is quite hard to characterize what is known about the software when a Cleanroom project is initiated. In some cases a great deal is known and in other cases much less is known. In any event what is known must be assembled and the software project team must prepare and publish a specification for the software.

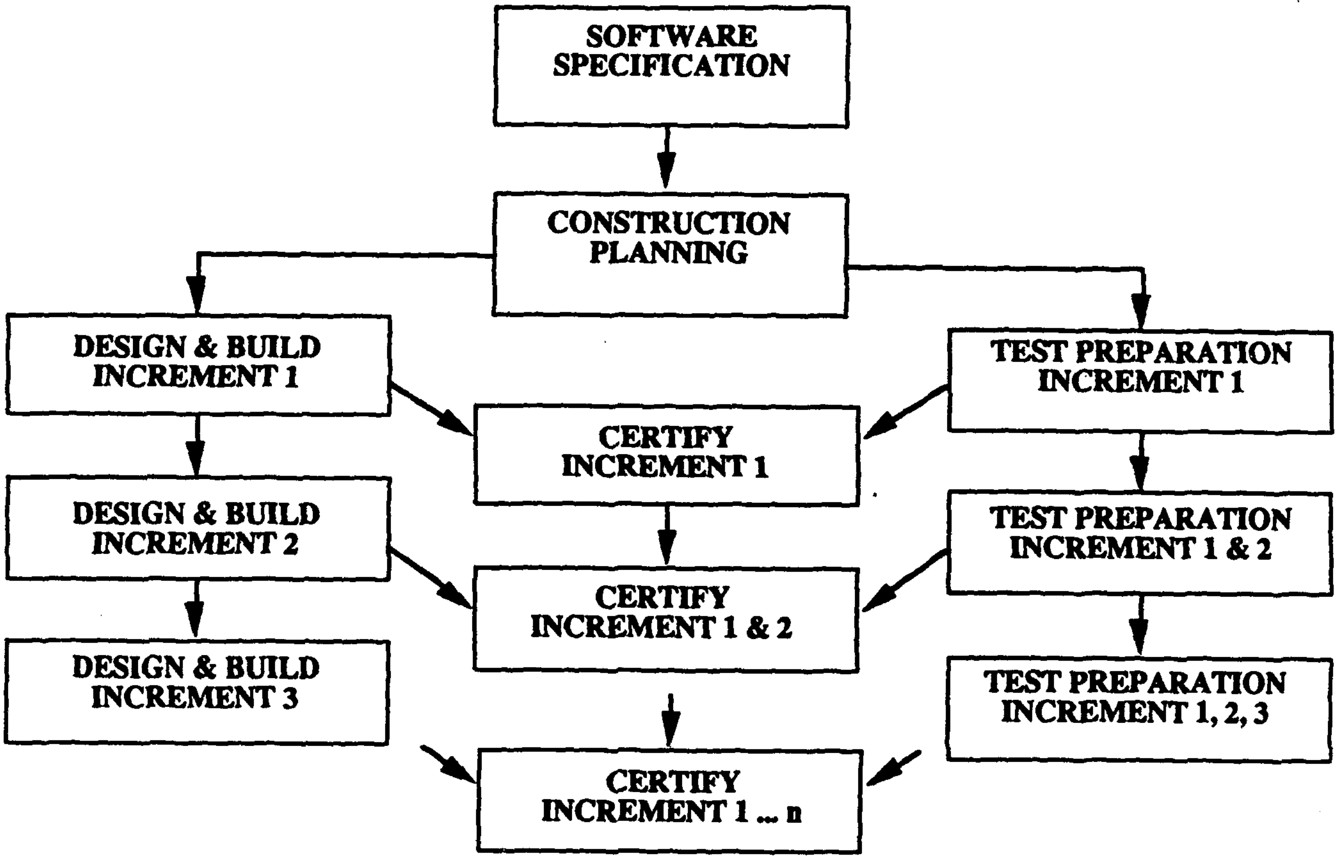
A Cleanroom specification is prepared in six volumes as follows:

Mission Volume
Defines the mission the software is to perform.

User's Reference Manual Volume
Defines the "look and feel" of the software from the user's perspective by defining the stimuli and responses the software consumes and produces.

Functional (Black Box) Specification Volume
Defines the black box responses produced by the software as a function of stimuli histories.

Figure 2.4.2.1 Typical Organization of a Cleanroom Project


SOFTWARE SPECIFICATION
CONSTRUCTION PLANNING
DESIGN & BUILD INCREMENT 1
DESIGN & BUILD INCREMENT 2
DESIGN & BUILD INCREMENT 3
CERTIFY INCREMENT 1
CERTIFY INCREMENT 1 & 2
CERTIFY INCREMENT 1 ... n
TEST PREPARATION INCREMENT 1
TEST PREPARATION INCREMENT 1 & 2
TEST PREPARATION INCREMENT 1, 2, 3

Functional Specification Verification Volume

Presents the justification as to why the Functional (Black Box) Specification is correct as written.

Expected Usage Profile Volume

Defines the statistical usage profile in each program state i for the software features including the probability p_{ij} that given the software is in program state i it will receive the stimuli necessary for the software to transfer to program state j.

Construction Plan Volume

Defines the sequence in which the software is to be constructed, in terms of increments.

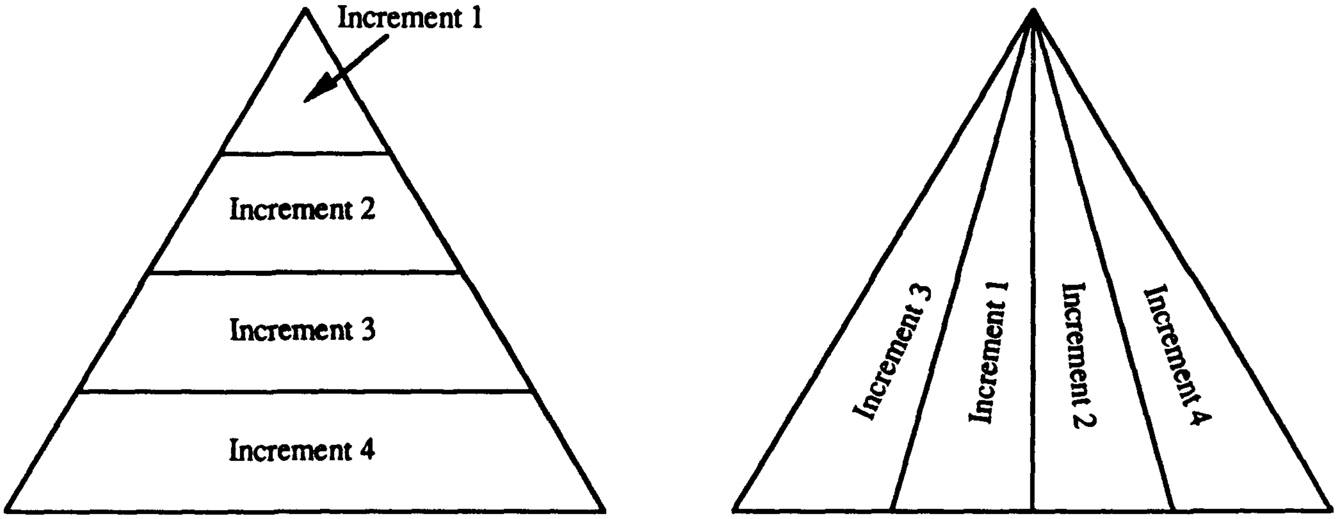
Specification volumes are developed in a spiral fashion since adding knowledge to one volume typically influences another volume, and each volume becomes successively more complete. Specification development continues until it is determined the specifications both represent what is wanted and that they are sufficient to precede. In a Cleanroom project the effort to produce the specification typically consumes 40 - 60% of the total project effort. A specification is always formal even if it is preliminary.

In parallel with developing the specifications it is necessary to engage in activities in order to understand the problem domain and to understand the solution domain. It is necessary to understand the problem domain to invent the stimuli and responses that maximize the effectiveness of the software when it becomes operational in the problem domain. It is necessary to explore the solution domain ahead of the design to determine the best route to developing the software to again maximize its effectiveness in the solution domain.

2.4.3 Developing a Construction Plan

Once the first four volumes of the specifications are accepted a plan for how to construct the software is prepared. The construction plan decomposes the entire software into increments. The individual increments accumulate into the entire software system. With Cleanroom Engineering, Increment 1 must be executable by user commands, Increment 1 plus Increment 2 must be executable by user commands and so on. It is traditional to develop software in increments but the requirement for user command executability of accumulating increments is a major difference. The fact that the accumulating increments are executable by user commands means that as increments are accumulated into the full system and the accumulating increments have user profiles and therefore, can be certified with usage testing. Figure 2.4.3.1 indicates the difference in construction strategy. The left side depicts a traditional top-down construction schedule where increments are defined as cross sections in the hierarchy. The right side depicts a construction plan that develops usage executable accumulating increments where the increments represent vertical slices of the usage hierarchy so user stimuli are transformed into user responses. Figure 2.4.3.2 shows the portion of the usage hierarchy that the accumulating system will represent for a 4-increment software project. Developing a software construction plan for user executable accumulating increments typically requires hard thinking.

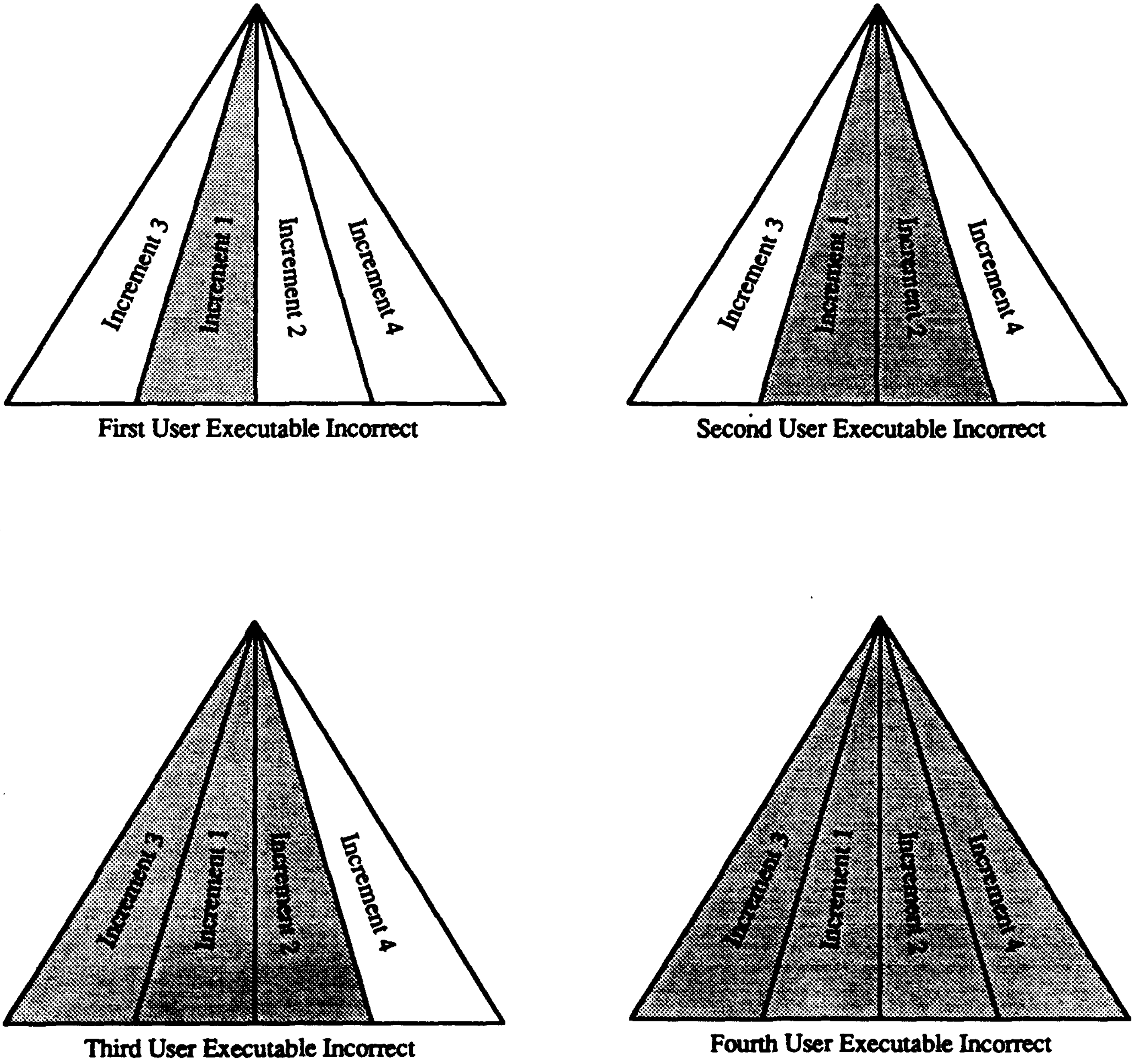
Figure 2.4.3.1 Difference Between Conventional and Cleanroom Incremental Construction Strategies



Conventional Top-down
Incremental Construction Plan

Cleanroom User Executable
Incremental Construction Plan

Figure 2.4.3.2 Cleanroom Incremental Testing Strategy Illustrated



First User Executable Incorrect

Second User Executable Incorrect

Third User Executable Incorrect

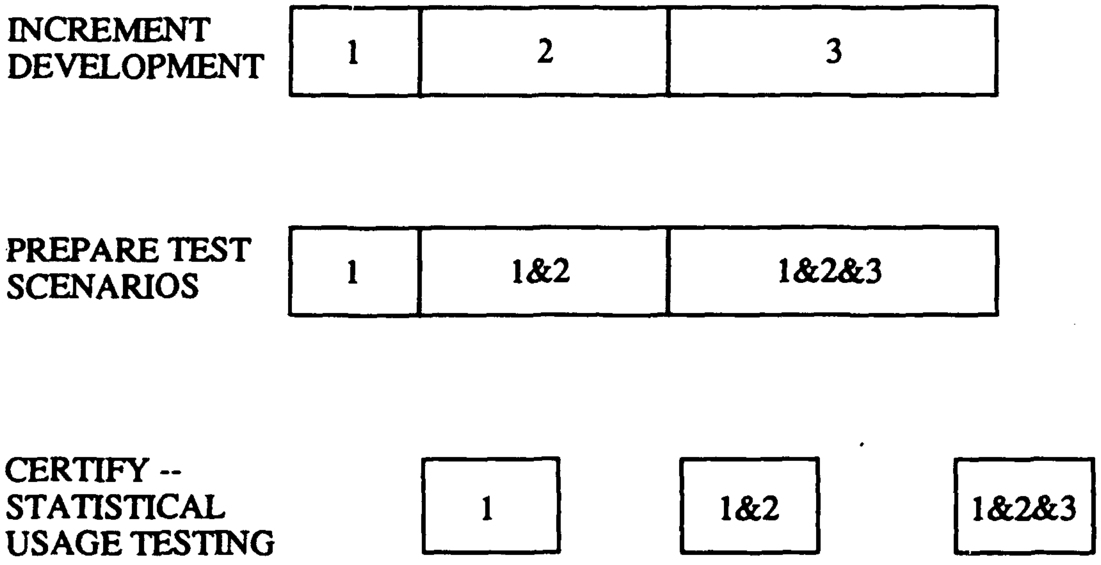
Fourth User Executable Incorrect

A sample construction plan is shown in Figure 2.4.3.3.

2.4.4 Developing And Certifying The Software

Following the adoption of a construction plan, the software is developed and certified by increment. The Development team develops Increment 1. In parallel the Certification team prepares test scenarios for Increment 1. When the Development team completes its verification of Increment 1 using functional

Figure 2.4.3.3 Typical Cleanroom Construction Plan



verification and other logical arguments the Development team turns the increment over to the Certification team for certification. The Development team does no compilation or execution testing of the increment. For Increment 2, the Development team develops, while the Certification team prepares test scenarios for Increment 2 and certifies Increment 1. This process continues for subsequent increments.

Once the Development team completes and submits the increment, the Certification team then compiles the increment. If any compilation failures are encountered the Certification team completes a failure report and returns the software to the Development team for correction. The increment is then corrected and returned to the Certification team which again compiles the increment. Assuming no compilation errors are found the Certification team assembles the system and begins certification testing by selecting a test case at random that was randomly generated and running it. All responses from each test case are examined for failures. Time to failure is recorded. The Certification team continues testing until the reliability targets are reached, or the volume of failures indicates that re-design is necessary. The Certification team enters the test results in the certification model data base and estimates the reliability of the software.

The Certification team then returns the failure reports to the Development team who then correct the failures. For each correction the Development team completes an Engineering Change Notice. When the Development team is satisfied that the increment contains no more failures the increment is once again returned to the Certification team. The Certification team resumes testing. Tests previously run are re-run to determine if previously observed failures have been fixed. These tests that are re-run are not used in the certification model, just new test scenarios.

The certification process continues until either the increment is certified to the desired level or it is determined the increment is so failure-laden that continued certification is unproductive and the best course is to scrap the increment and develop a new increment. Typically in the Cleanroom environment, failures are encountered very early in the test scenario as the simple errors that the logical verification process leave in the software are uncovered. Very soon the few simple errors the Development team left in the software are found and test cases begin long runs before failures are encountered. It is typical that only a few cycles of corrections are required before an increment can be certified to the desired level of reliability.

2.4.5 Human Factors

Previous sections have emphasized the technology. The human factors of Cleanroom Engineering are just as important. The goal of the Cleanroom technologies is to provide software developers with the tools and practices they require for the job they are being asked to do. Software developers want to do a good job. They know the users of their products do not like to experience failures. They know their organizations would like it much better if their software was not failure-laden. They know their managers are asking them to produce quality software. They also know that their managers are not giving them the resources that permit them to deliver on their own desires or on management's desire to produce high-quality software. This is a recipe for low morale and discontent.

The first thing that Cleanroom does for software developers is that it provides them the means to deliver quality. This is a sure path to higher morale and increased job satisfaction. In recent years software developers are being referred to as "software engineers." In fact it is widely suspected that software developers are engineers in name only when they are using contemporary software development practices. So the second thing Cleanroom does for software developers is that it provides them with the means to become software engineers both in deed and name.

Other aspects of the human side of Cleanroom Engineering include:

Emphasis on eliminating rework - get it right the first time so it is not necessary to track and correct an error on previously completed work.

Emphasis on team effort - the team is responsible for quality.

Assignment of responsibilities to teams based upon division of concerns.

Emphasis on giving people the rigorous practices they require to specify, develop and certify software.

Emphasis on thinking, justifying and verifying prior to doing.

Emphasis on specifying before implementing; that is, do not start coding early under the assumption that there will be a lot of debugging to do.

2.5 References

[1] Adams, E. N., "Optimizing preventive service of software products," IBM Journal of Research and Development, January 1984.

[2] Basili, Victor R., "Software Development: A Paradigm for the Future," UMIACS-TR-89-57, College Park MD, University of Maryland, June 1989.

[3] Cho, C., Quality Programming: Developing and Testing Software with Statistical Quality Control, John Wiley and Sons, New York, 1987.

[4] Cobb, R. H. and H. D. Mills, "Engineering software under statistical quality control," IEEE Software, November 1990.

[5] Currit, P. A., M. Dyer and H. D. Mills, "Certifying the reliability of software," IEEE Transactions on Software Engineering, Vol. SE-12, No. 1, January 1986, pp 3-11.

[6] Green, S.E., A. Kouchakdjian, and V. R. Basili, "The Cleanroom case study in the software engineering laboratory: project description and early analysis," Software Engineering Laboratory, SEL-90-002, March 1990.

[7] Linger, R. C. and H. D. Mills, "A case study in Cleanroom software engineering: The IBM COBOL structuring facility," Proceedings of COMPSAC '88, IEEE, 1988.

[8] Linger, R. C., H. D. Mills and B. I. Witt, Structured Programming: Theory and Practice, Addison Wesley, 1979.

[9] Mills, H. D., "The new math of computer programming," Communications of the ACM, Vol. 18, No. 1, 1975.

[10] Mills, H. D., Software Productivity, Little, Brown and Company, 1983.

[11] Mills, H. D., "Structured programming: Retrospect and prospect," IEEE Software, November 1986, pp 58-66.

[12] Mills, H. D., "Stepwise refinement and verification in box-structured systems," IEEE Computer, June 1988, pp 23-36.

[13] Mills, H. D., V. R. Basili, J. D. Gannon and R. G. Hamlet, Principles of Computer Programming: A Mathematical Approach, Wm. C. Brown, 1987.

[14] Mills, H. D., M. Dyer and R. C. Linger, "Cleanroom software engineering," IEEE Software, September 1987, pp 19-24.

[15] Mills, H. D. and R. C. Linger, "Data structured programming: Program design without arrays and pointers," IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, February 1986, pp 192-197.

[16] Mills, H. D., R. C. Linger and A. R. Hevner, Principles of Information Systems Analysis and Design, Academic Press, 1986.

[17] Mills, H. D., R. C. Linger and A. R. Hevner, "Box structured information systems," IBM Systems Journal, Vol. 26, No. 4, 1987, pp 395-413.

[18] Mills, H. D. and J. H. Poore, "Bringing software under statistical quality control," Quality Progress, November 1988, pp 52-55.

[19] Musa, J. D., A. Iannino, K. Okumoto, Software Reliability: Measurement, Predication, Application, McGraw-Hill, New York, 1987.

[20] Musa, J. D., W. E. Everett, "Software-Reliability Engineering: Technology for the 1990s," IEEE Software, November 1990.

[21] Parnas, D. L., "A technique for software module specification with examples", Communications of the ACM Vol. 15, No. 5, May 1972, pp 330-336.

[22] Parnas, D. L., "Designing software for ease of extension and contraction," IEEE Transactions on Software Engineering, SE-5, No 3, March 1979, pp 128-138.

[23] Parnas, D. L. and Y. Wang, "The Trace assertion method of module-interface specification", Tech. Rep. 89-261, Queen's University, TRIO, October 1989.

[24] Selby, R. W., V. R. Basili and F. T. Baker, "Cleanroom software development: An empirical evaluation," IEEE Transactions on Software Engineering, Vol. SE-13, No. 9, September 1987.

[25] Wirth, N., "Program development by stepwise refinement", Communications of the ACM Vol. 14, No. 4, April 1971, pp 221-227.

Section 3
A Box Structures Model for Cleanroom Engineering

In this section, the processes used to develop software in a Cleanroom Engineering environment are specified using Box Structures. It assumes an understanding of the general Cleanroom Engineering ideas and practices and of the Box Structures approach to modeling a process.

3.1 Process Models

Process models are briefly introduced before developing the Cleanroom Engineering process model. A **process** is a set of progressive steps which transform stimuli into responses. A **software development process** is then the set of steps that people use to prepare software to satisfy a defined need. There is a wide divergence between organizations in how well the process they use to develop software is defined and if a process is defined how well it is followed.

Software development is a complex process. Complex processes are documented by developing an abstraction of the process called a model. Models need to be simple enough that people are able to use the model to reason about real world occurrences while at the same time sufficiently realistic so the reasoning produces sound extrapolations to the real world.

Researchers who have been modeling the software development process have been finding it difficult to develop a model that reaches the right combination of simplicity and realism. The reason is that it is very difficult to develop a simple but yet realistic model of a process when that process is not under the complete intellectual control of those people who are performing the process. It is not surprising that software development is not in a state of intellectual control since society has been developing software for less than a human generation. Software development is still largely a heuristic process where software developers make large inventions with little verification of each invention.

Cleanroom Engineering practices permit engineers to develop and verify software using procedures which a have a sound mathematical basis. The procedures establish intellectual control by giving engineers the ability to make an orderly sequence of small inventions following each invention with a verification that enables the engineer to quickly find faults in their reasoning. The result is that it is has been possible to develop a useful model of the software development process.

3.1.1 Box Structures of Processes

Processes exist in real time since stimuli arrive and responses are produced in real time. Due to the complex nature of the process it is useful to consider three different descriptions of a process.

There is the black box description which is a function which maps the histories of the stimuli which impact the process into responses. Black box descriptions are implementation free. They define the inputs to and the products of the process.

Processes typically store data between stimuli in order to respond to the effect of prior stimuli. A state box description is a function which maps current stimuli and current state data (data known at this level with the effects of hidden data defined in lower parts) into a response and new state.

Processes typically use lower level processes to perform an allocated process. A clear box description expands the state box description in terms of black boxes or processes to use at the next lower level in the hierarchy. Process decomposition continues until the process is fully defined without the need to any new process in any clear box. Experience with structured programming leads us the definition of four types of clear boxes: sequence, alternation, iteration and concurrent.

Experience indicates that people are able to develop much better models of complex processes like software development when they examine the process from three perspectives (Black Box, State Box and Clear Box). A common difficulty with models (including many of the process models that have been developed) is that the functions defining the model are a combination of the three functions.

Processes are often quite complex. As a result a process is typically decomposed into child processes and the child processes are further decomposed into more child processes. The using relationship between processes is often represented in a usage hierarchy that defines which child processes are used by each parent process.

In the process model as developed a hierarchy of processes are considered where the parent process is "develop software". The parent process is subdivided into processes. The division continues until all the leaf nodes require no further decomposition.

3.1.2 Components Of A Software Development Process Model

A process model to be complete must define the following:

what-to-do	the sequences in which processes must be developed
how-to-do-it	the way to perform the process
when-it-is-done	the conditions that must be satisfied to say the process has been completed correctly
where-to-put-it	the state data to update in order to reflect the results of the process
measuring how well it has been done	quality and productivity metrics are required to support process improvement

Each of these components of the model is briefly discussed in the following paragraphs.

3.1.3 Realizations of Process Models (or what-to-do)

Each software development project will result in a realization. A realization is a usage hierarchy of the specific processes that need to be performed to complete the given software development project. The realization can be developed by specializing the process model with the details of the actual software development. From the usage hierarchy it is possible to determine all the feasible sequences in which the processes can be performed. Project managers must determine which of the possible sequences of processes will result in the best utilization of resources.

As the project progresses changes will occur, new information will be obtained, new ways to minimize risk will be uncovered, etc. Each of these changes may cause the addition of new processes or the deletion of processes from the project usage hierarchy. The result will be a revised sequence. In this way the realization of any project is a living plan which dynamically changes to reflect the actual site of the project.

3.1.4 Engineering Practices (or how-to-do-it)

The definition of the how-to-do-it steps is sometimes referred to as the methodology for performing the named process. The term engineering practice seems to be a better term to use when referring to the how-to-do-it steps for a process. The reason is that the term engineering practice is used to denote an orderly procedure that is based upon sound scientific principles while methodology is used to denote any orderly procedure. Therefore, by using the term engineering practice the focus is on eliminating all procedures that are not based on sound scientific principles. The elements of an engineering practice are engineering tasks.

3.1.5 Completion Conditions (or when-it-is-done)

In the process model of software development each process has a **dountil** structure. The **until** part is a set of completion conditions. Completion conditions are predicates that engineers use to determine if the process is complete. Completion conditions examine the results of the process as recorded in the state data. For leaf nodes in the usage hierarchy the **do** part of the process is composed entirely of engineering tasks. These engineering tasks for the development and certification portions of the project cycle are defined rigorously by the Cleanroom engineering practices. For the specification portion of the life cycle the engineering tasks are not yet defined rigorously since no one has yet defined a set of rigorous engineering practices that someone can follow in order to develop a specification. For intermediate nodes of the usage hierarchy the **do** part of the process is composed of a control structure defining which subprocess should be invoked.

3.1.6 State Data (or where-to-put-it)

The view point of the model is that engineers perform processes by referencing files that contain the state data which define the current state of the project. At each stage the engineers find the information they require in the state data files, use their intellect guided by the Cleanroom Engineering practices to make a next invention and then record the invention along with its rationale, and where applicable, a verification in the new state. This process continues until the design is complete. Engineers always make small inventions followed by verifications where possible, until the completed software is available. When the software is complete and certified the state is used to produce the final responses in terms of distribution software, maintenance software, final specification, user documentation, design documentation and project metrics. During the course of the project the project team needs to produce intermediate results that are used by people out side of the project to access progress and the adequacy of the design to determine if the software will meet its design objectives. The intermediate results need to be extracted from the state data files as they then exist.

The state data then plays an essential role in a software project. It is vital that the project team and all other interested parties agree on a format for the files that contain the state data and that the project team be meticulous in maintaining all project information in the state data files in the prescribed form. In this way the everyone associated with the project can proceed with confidence.

A good analogy is the financial records of a company. The reason accountants can produce financial accounts that everyone generally accepts as representing the current state of the financial progress of the company is that they meticulously maintain files containing the state data that reflect all the transactions and they then follow processes to extract the required information for some report. The process works because it is orderly and followed meticulously. The same can be true for software projects. The significant difference between accounting and software development is that in accounting the transactions that are tracked in the state files come from outside the accounting department while in a software project the transactions reflected in the state files result from inventions made be the software engineers. But the need for meticulous record keeping is the same.

3.1.7 Measuring Quality and Productivity (or measuring how well it has been done)

Organizations desire to organize the process by which software is developed so there is a high probability that the software will: (1) provide its users with high quality service, and (2) be developed with the minimum possible investment. Organizations require measurements to determine how well they are succeeding in accomplishing their objectives. If people do not measurements they are flying blind and have no idea how well they are doing against overall objectives or how different teams are performing relative to each other. Organizations find it relatively easy to make gross measurements of productivity but using conventional software development practices they find it exceedingly difficult to measure the quality of the software. They know they have produced 10,000 lines of code with 70 staff months of effort. The question is how many latent failures remain in the software. If they are many latent failures much more effort will be required. The problem is how much more effort.

One of major difficulties being faced by researchers trying to model conventional software development practices is what pseudo measures to use to determine the quality level of the software as it is being developed. This is not a problem for Cleanroom development since development occurs in a pipeline of user executable increments and following the addition of each increment to the accumulating software accurate projections of the quality of the software as developed are made. The measurements result in accurate estimates of the Mean Time To Failure (MTTF) for the software as it then exists. As a result engineering managers have a continuing measurement of the quality of the development process for which they are responsible. In this way they can take prompt corrective action whenever the process begins to slip out of control. This feature greatly simplifies the actual development of software. It makes it much easier to develop a model of the software development process.

3.1.8 Using A Process Model

In creating a process model the goal is to develop an abstraction of reality that is sufficiently complete and accurate so that one can use the model to reason about real software development projects which are complex and messy. On the other hand the abstraction should be small and simple enough so the people are able to use it conveniently to reason about real software development projects. Process models are used in several different ways to help engineers and managers develop software. These include:

> Project Planning. A project plan is defined by a usage hierarchy of all the processes required to complete a definite software development project. This can be accomplished by expanding the model usage hierarchy into a usage hierarchy for a definite software development project. The precise expansion will depend on the project. There are as many different realizations of usage hierarchies possible as there are software projects.

Project Scheduling. Given a project plan it is necessary to prepare a schedule which allocates available resources to the project in accordance with the project plan. Given a plan in the form of a usage hierarchy, estimating metrics for each process and a definition of available resources it is easy to prepare a project schedule. Expenditure of effort and reported progress toward completing the process can be recorded against the schedule. The schedule provides a convent way for engineers and managers to communicate about expectations and accomplishments.

Engineering Management. Engineers who are performing a project require day-to-day guidance as they perform an assigned process. A precise description of how to perform a process and what needs to be shown to know the process has been completed satisfactorily are what engineers need to know in order to manage their work on a daily basis. A process model should provide engineers and engineering management with the ability to know what to do and communicate with each other about current status of a process.

Project Monitoring. The successful performance of any process requires the ability to measure the progress being made toward completing the process. One of principal difficulties with software development as practiced today is obtaining accurate measures of progress in terms of quality. The lack of accurate quality measures forces people to define pseudo measures. This complicates both actual practice and models of the process. A big advantage of Cleanroom is that the engineering practices being followed provide frequent and accurate measurements of software quality. Therefore, the process model for Cleanroom is less complicated as are actual projects.

Process Improvement. An important feature of any operation is that the operation requires a built in way to constantly monitor operations and then improve the operation on a continuing basis. A prerequisite to evolutionary development is to have a model of the processes and practices being followed. In this way it is possible for management to reason about what changes to make in the current process in order to maximize the return on the investment in process improvement and to then make measurements in order to determine if the targeted results were realized.

3.2 A Guide To The Process Model

The balance of this section is organized into eight sections as follows. Sections 3.3 and 3.4 describe the stimuli that the engineering team receives as they design and develop the software and the responses they are required to produce during the project and at the completion of the project. The engineering team receives six stimuli and must deliver thirteen responses. There is nothing unique about these stimuli and responses for a Cleanroom Engineering project. These stimuli and responses are the same for all software development projects no matter what processes and engineering practices are being used to define the software. In Section 3.5 the first level Black Box is described. The functions which describe how engineers produce responses in terms of stimuli histories are what distinguish one software development methodology from another. Therefore, definition of the Cleanroom Engineering process is begun in this section.

Section 3.6 describes the state data that engineers need to maintain as they process the stimuli. The state data are inventions which are made to represent the cumulative effect of all the stimuli received - that is, the stimuli histories. Eight classes of state data have been invented. In Section 3.7, the first level State Box transformation is introduced. In Section 3.8, the first level Clear Box is described and then decomposed into 6 second level Black Boxes which are in turn decomposed into 18 third level Black Boxes in Section 3.9. A total of 25 Black Boxes then define the processes that make up a Cleanroom Engineering project. Each process is described in terms of pseudocode or text and also has a graphical representation in terms of Box Structures. Section 3.10 addresses the issue of reviews and how they are perceived in the Cleanroom process. Section 3.11 discusses some of the differences between this model and other software development models. Section 3.12 summarizes how the model can be used.

These 25 processes define a generic work break-down structure which can then be used by engineering team management to develop a specific project plan. The form of each leaf process Clear Box is an iteration Clear Box where the completion criteria for the process are the predicates which define when the iteration loop is complete. This style of representation works very nicely to define a model which is simple to comprehend and use to plan and control projects but which is sufficiently complete to model even the most complex software development process. The use of formal procedures to develop the project cycle for Cleanroom Engineering has produced useful results. Section 4 provides detailed descriptions of the 25 processes.

The model as developed defines tasks performed by Cleanroom Engineering teams and first line managers. These engineering teams interface with many different groups, all of whom have an interest in the successful conclusion of the project. These groups typically include the line management of the organization to which the engineering team belongs, members of the staff organizations that advise the line managers and representatives of the customer's organization that is buying or contracting for the software. The groups that participate and the extent of their participation depend on the situation. In the following pages, these diverse groups are sometimes referred to by the abstraction "Management." Management, as represented by this abstraction, takes on several roles: it is the engineering group's client, it is the consumer of the software, it establishes and changes requirements, it provides resources, it measures progress, it establishes rewards and punishment for good and poor work, and it accepts and rejects the completed software.

In balance of this report, the following notation is used to represent various objects involved in the model as follows:

Si to represent Stimuli i.
Ri to represent Response i.
Ti to represent sTate data i. (T was used since S was already being used and T follows S.)
Ei to represent Engineering process i.
Ci to represent Condition i.
Ii to represent Internal stimuli i and response i that represent messages passed between processes.

3.3 Stimuli To First Level Cleanroom Engineering Black Box

The engineering team receives stimuli as they design and develop the software. There is very little the engineering team can do to influence the form in which stimuli arrive. They must accept and process the stimuli guided by their intellect the best way possible as they are received. The six stimuli they receive are:

S1 Project Charter

The Project Charter defines the instructions that the engineering team receives to undertake the software development. The Project Charter either directly or implicitly defines the scope of the effort, what is expected, the target schedule and budget, references to what is known about the situation and plans for software automation, and provides the authorization to the engineering team to expend effort. The Project Charter should be amended as required in order to modify instructions to the engineering team.

S2 Documents/Working Papers

Documents/Working Papers represent all written material, excluding the Project Charter, Project Review Minutes, and Master Project Plan, that is given to the engineering team from outside organizations. Typically, much thought and effort precedes the decision to undertake a software development project, and much effort is expended to gather source material. Typically, written material continues as the project progresses. These writings have many formats and contain vital, as well as irrelevant or misleading, information. Often related efforts to design hardware, manual processes, etc., proceed in parallel with the software engineering effort and result in a continuing series of documents.

S3 Informal Communication

In addition to the Documents/Working Papers received from people outside the engineering team, the engineering team receives information informally (much of it orally) from outside groups. This stimulus is called Informal Communication. The stimulus represents any solicited or unsolicited informal information given to the Specification, Development and/or Certification teams by any outside organization. This material may or may not be useful to the engineering team.

S4 Project Review Minutes

During the course of a project, the engineering team participates in Project Reviews. A Project Review is a formal meeting where people outside of the engineering team review one or more aspects of the proposed software solution. The result of these reviews are suggestions, recommendations and information. These results should be documented in Project Review Minutes by the review team.

S5 Resource Allocation

The knowledge team members possess in the application and solution domains have a direct bearing on the success or failure of the project. The combined team experience and knowledge is a stimulus to the project. Additionally, computers, software tools and other items that have an effect on the specification, development or certification of the project are considered a part of this stimulus.

S6 Master Project Schedule

Along with the Project Charter (sometimes subsequent to it) the engineering team is given a Master Project Schedule which defines dates of system-wide reviews, as well as start and end dates for the entire project. A schedule for the submission of status reports and project reviews may also be included. This schedule will later be updated by information received from the software engineering team to include intermediate review and build schedules.

In summary, software project **stimuli** are:

S1:	Project Charter
S2:	Documents/Working Papers
S3:	Informal Communication
S4:	Project Review Minutes
S5:	Resource Allocation
S6:	Master Project Schedule

Readers will note that project requirements have not been listed as stimuli to the software project. The requirements are, of course, stimuli to the system. But requirements appear in a number of ways. Ideally, many of them are listed in the Project Charter. Many may also appear as Documents or Working Papers. There is often a report that includes "Software Requirements" in its title. Other requirements arrive as a result of Informal Communication. The bottom line is that the requirements are stimuli, but are not a unique or separate stimulus, since they almost always appear in a variety of forms.

3.4 Responses From First Level Cleanroom Engineering Black Box

The engineering team provides responses to people outside the engineering project both during the project and at the completion of the project. The responses that are the objective of the project are produced at the completion of the project. There are five responses. There are eight responses that the engineering team produces during the course of the project. The thirteen responses are summarized in subsequent paragraphs. An additional discussion of Cleanroom Engineering responses appears in Section 6.

R1 Distribution Software
R2 Maintenance Software

The software as designed, implemented and certified as meeting requirements can be regarded as the primary response of the Cleanroom Engineering project. This software is packaged in two forms. The first is the software ready for distribution — the Distribution Software. The second is the software ready for the organization responsible for archiving and maintaining the software — the Maintenance Software.

R3 Final Specification
R4 User Documentation
R5 Archived Documentation

The software must be documented in several forms as follows:

1. The Final Specification defines what the software does from both the user's standpoint and the computer's vantage point and the usage profile which the software has been designed and certified to satisfy.

2. The User Documentation which is employed by users to help them use the software. The format and extent of User Documentation depends on the requirements. Typically it may include reference manuals, tutorial manuals, education material and marketing documentation.

2. The Archived Documentation represents the final state of the Cleanroom Engineering development process when the software is complete and certified. From one point of view, software development can be regarded as developing the state during the entire software solution specification, design, implementation and certification process and then once the software is complete the converting of the state into the responses as defined above. The Archived Documentation is a permanent set of documentation that represents the final state. The extent and formality of the Archived Documentation is specified by the requirements.

R6 Project Review Documentation

Project reviews are meetings between the engineering team and one or more external organizations. engineering teams prepare documentation for use at project reviews. Project reviews are typically held on projects at critical points in the life cycle so project sponsors and/or management can assess progress and the degree to which the software as it is being designed meets the needs which are to be satisfied by the software. Additionally, project reviews can be called by project management when they see a need, or on a schedule, such as a monthly status review. The exact nature of the documentation that is produced for these reviews typically depends on the nature of the development contract or the customs of the organization. For example, if the contract is being developed under a government contract, the reviews may be specified by 2167 or 2167A. Other organizations have their own specifications for project review documentation. This project review documentation must be produced at the specified time in the specified format from the state data. This documentation is produced at many different points in the project life cycle. In many cases, the development organization is free to propose a format for Project Review Documentation.

R7 Project Schedule

The Master Project Schedule is updated with schedule information received from the Cleanroom Engineering team. The Project Schedule is the response which provides this information. The schedule is a published description of the schedule of milestones and deliverables between the engineering teams and management. The dates given in the Project Schedule commit the engineering team to a schedule for production of the system, to which "management" can hold the engineering team accountable.

R8 Management Metrics

Measurements of engineering processes are collected routinely. The measurements are used by management to control the development and to study development performance in order to find improvements for future projects. Individual managements will want to collect different information; therefore, the exact metrics collected will differ from project to project. These measurements as defined for permanent archiving are referred to as Management Metrics.

R9 Project Status Reports

Project Status Reports are distributed to management based on the schedule dictated in the Master Project Schedule. Information in the reports allows management to assess the status of the project regarding budget and resources, and make modifications to the project as necessary.

R10 Externally Submitted Questions or Issues

During a project, there are often questions or issues that cannot be answered by members of the Certification, Development or Specification teams. These questions or issues are submitted to organizations outside the engineering task in order to be resolved. The issues are resolved, with their response being either Documents/Working Papers or Informal Communication.

R11 Schedule Change Request

During a project, the engineering team may determine that a milestone or deliverable may not be completed on schedule. For these types of situations, the engineering team must have the opportunity to request a schedule modification. This request will prompt management to analyze the issue and determine whether to grant a schedule change. If the change is accepted, the engineering team will receive a new Master Project Schedule as a stimulus.

R12 Suspend Project Pending Management Decision

During a project, the case may arise where the engineering team believes that the project must be suspended. For example, the team may determine that the code does not meet the specifications, or that the specifications do not meet the requirements. Since management has the final decision as to whether the project should be suspended, this response is a prompt to management to closely assess the project and make a decision, as to how the project will continue, or whether the project should be reorganized or cancelled.

R13 Project Reports

During a project, the Specification, Development or Certification teams may produce a document that has use outside of the engineering team. These documents are published outside the engineering team, so other individuals in the organization can use the knowledge gained by those members of the engineering team.

In some cases, a project sponsor will decide that some other responses are necessary, in addition to the ones that appear in this section. These additional responses are not addressed because it would be impossible to identify the full range of potential responses. The responses discussed are a fundamental set of project responses. Some additional responses can be accounted for by viewing them as material that must be a part of a review. As reviewed material, it will be delivered to the sponsor. For other responses, it may be necessary to tailor this document for the specific organization and project in order to account for such requests.

In summary, software project **responses** are:

R1: Distribution Software
R2: Maintenance Software
R3: Final Specification
R4: User Documentation
R5: Archived Documentation
R6: Project Review Documentation

R7: Project Schedule
R8: Management Metrics
R9: Project Status Reports
R10: Externally Submitted Questions or Issues
R11: Schedule Change Request
R12: Suspend Project Pending Management Decision
R13: Project Reports

3.5 First Level Cleanroom Engineering Black Box

First Level Black Box Transformation

The stimuli are analyzed and processed by the Cleanroom Engineering team using Cleanroom Engineering ideas and practices and the team's accumulated experience and intellect. The team develops the responses proceeding top down with small step inventions guided by bottom-up explorations to increase understanding. The stimuli define a system to be developed. The Cleanroom Engineering model defines a set of iterative processes by which the stimuli are understood and the responses are designed, verified, implemented and certified. If the Cleanroom Engineering ideas are followed, the inventive process will proceed in an orderly manner under intellectual control. As a result, the process has the following very desirable feature: if software is developed by different teams (which are competent in Cleanroom Engineering practices) from the same specification, the resulting software solution will be very similar. The first level software engineering Black Box is graphically described in Figure 3.5.1.

3.6 State Data: A Summary

During the software development project, the project team studies and analyzes the data contained in the **stimuli** and information derived from the **stimuli**, and develops the required **responses** using Cleanroom Engineering concepts and practices. The team records the insights gained from many individual analyses and the results of many small inventions and invention verifications in files which contain the **state** of the software. These files represent the **state** of the software design and implementation at any point in time during the inventive process. These files may be electronic or paper.

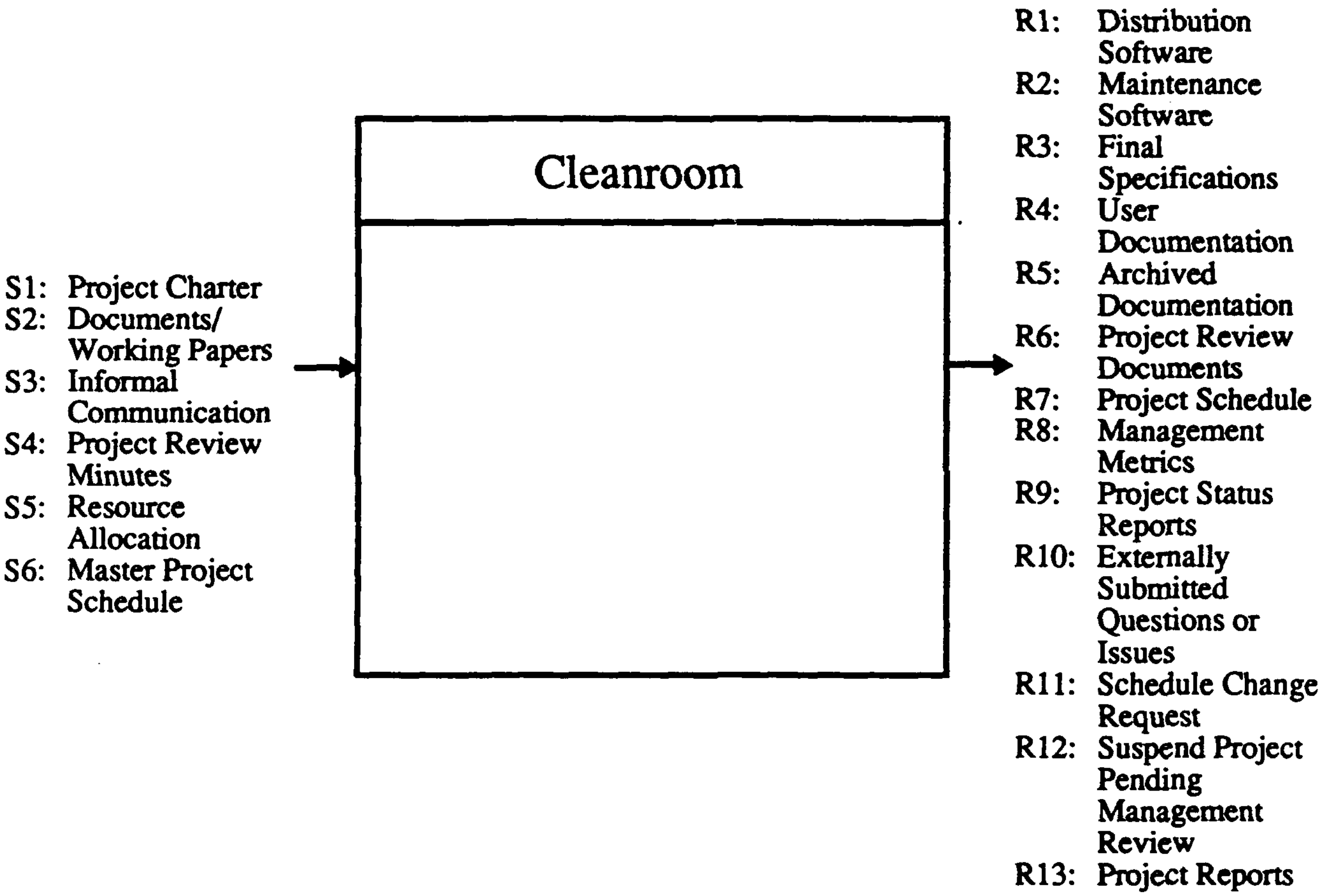
Engineers performing each of the Cleanroom Engineering processes that comprise a software development project work primarily with the files containing the state data which define the current state of the project. At each stage, the engineers find the information they require in the state data files; use their intellect guided by Cleanroom Engineering processes to make a next invention; and record the invention along with its rationale (and where applicable, a verification) in the new state. This process continues until the design is complete. The engineers always make small inventions followed by verifications where possible, until the completed software is available. When the software is complete and certified, the state is used to produce the final responses in terms of distribution software, maintenance software, final specification, user documentation, archived documentation and project metrics.

During the course of the project, the team needs to produce intermediate results that are used by people outside the project to assess progress and the adequacy of the design to determine if the software will fulfill its assigned mission. The intermediate results need to be extracted from the state data files as they then exist.

The state data then plays an essential role in a software project. It is vital that the project team and all other interested parties agree on a format for the files that contain the state data and that the project team be meticulous in maintaining all project information in the state data files in the prescribed form. In this way everyone associated with the project can proceed with confidence.

All software project state data can be maintained in eight files summarized in this section. A more complete description of each of the state data files is presented in Section 5.

Figure 3.5.1 Black Box for Cleanroom


Cleanroom
S1: Project Charter
S2: Documents/ Working Papers
S3: Informal Communication
S4: Project Review Minutes
S5: Resource Allocation
S6: Master Project Schedule
R1: Distribution Software
R2: Maintenance Software
R3: Final Specifications
R4: User Documentation
R5: Archived Documentation
R6: Project Review Documents
R7: Project Schedule
R8: Management Metrics
R9: Project Status Reports
R10: Externally Submitted Questions or Issues
R11: Schedule Change Request
R12: Suspend Project Pending Management Review
R13: Project Reports

The viewpoint taken by this model that all intermediate results of engineering processes are state data makes the model much easier to develop and use. When the actual data produced on projects is considered state data, it is much easier to manage and perform on software development projects since it moves the data out in the open and makes it public to all members of the engineering teams. This is in contrast to other abstractions that continue to regard project state, other than deliverables, as private data that if ever available is only regarded as an artifact.

T1 Project Document Files

The project document file contains a library of documents that are relevant to the project. The library contains six types of documents:

1. a Project Document Index which describes the organization of the contents of the files,
2. External References which are documents prepared by groups outside the project,
3. Project Reports which are documents that have been prepared by some project team member and published outside of the project,
4. Working Papers which are internal documents prepared by some project team member that are internal to the project team,
5. Trade Studies which evaluate the relative value between alternatives using a quantitative comparison scheme, and
6. Project Review Minutes which define conclusions and action items from project review meetings.

This material is available to all members of the engineering team for use as reference documents. The file should be the archive of all externally received material, as well as internal work that is not limited to a specific team. More detailed descriptions of the Project Document Files are given in Sections 5.0 and 5.1.

T2 Software Specification Files

The Software Specification files contain the official copy of the specifications for the software. This file is maintained in electronic form using a desk top publishing system chosen by the project team. The specifications are in six volumes as follows:

1. the Mission Volume which defines the mission the software is to perform,
2. the Users Reference Manual Volume which defines all the external inventions (stimuli and responses) which define the software from its user's perspective,
3. the Black Box Function Volume which defines an implementation free internal view of the software in terms of Black Box functions,
4. the Black Box Function Verification Volume which justifies the correctness of the Black Box Function Volume,
5. the Usage Profile Volume which defines the expected usage profile of the software by its users in the planned usage domain, and
6. the Construction Plan which presents the schedule for deliverables and milestones that the engineering team will follow.

The Software Specifications are developed using a spiral workplan. More detailed descriptions of the Project Document Files are given in Sections 5.0 and 5.2.

T3 Software Development Files

The Software Development files contain documents that are relevant to the Development team. The file contains eight types of documents:

1. a Design Index which defines all references to design objects.
2. Trade Studies that evaluate the relative value between alternatives using a quantitative comparison scheme,
3. Boxes that define the Black, State and Clear Box transition formulas for each box in the usage hierarchy,
4. Verifications which record the verification arguments for box expansions and code expansions,
5. Code Refinements that define the expansions of Clear Box functions into code,
6. Design Notes that record any information that the designer feels should be preserved,
7. Code, and
8. Metrics.

These eight categories are sufficient to serve as a referencing and archiving schema of the entire Development team process. More detailed descriptions of the Software Document Files are given in Sections 5.0 and 5.3.

T4 Software Certification Files

The Software Certification files contain documents that are relevant to the Certification team. The file contains eight types of documents:

1. a Certification Index which describes the organization of the remainder of the file,
2. Certification Notes which are used to preserve any issues pertaining to certification,
3. the Sampling Plan which defines how to estimate the reliability of the software to be certified,
4. Script Generators/Test Scripts which are scripts or code that define the next input for each test case,
5. Test Scenarios which define the test cases to be run,
6. Solutions that are the expected output for each test case,
7. Results which record what was produced by a test scenario for a version of code, including a pass/fail determination, and
8. Metrics.

This information is sufficient for the Certification team to complete all of their processes and for later archiving of their work. More detailed descriptions of the Software Certification Files are given in Sections 5.0 and 5.4.

T5 Project Management Files

The Project Management Files contain documents that are necessary to manage and control a project. This file contains six types of documents:

1. the Project Charter,
2. Master Project Schedule,
3. Engineer Activity Records which are all status reports submitted outside of the engineering team,
4. the Project Schedule,
5. all Task Assignments, and
6. Metrics.

This information should be sufficient to support decision making, or the review of project status by engineering team members. More detailed description of the Management Files are given in Sections 5.0 and 5.5.

T6 Unresolved Questions or Issues

Each time the engineering team asks a question to others inside or outside the engineering team to gain more information, test a hypothesis, etc., the question is recorded in the unresolved question file. When the answer is received it is recorded in the same file. More detailed descriptions of the Unresolved Questions or Issues are given in Sections 5.0 and 5.6.

T7 Pre-Release Software

The Pre-Release Software file is the library of software that has been turned over to the Certification team by the Development team for certification. This file is maintained by the Certification team using rigorous configuration management techniques. If the Development team requires a copy of some of the software to remedy an observed failure, they request a copy from the Certification team. The fix for the observed failure is returned to the Certification team by the Development team as an engineering change. The engineering change specifies the changes required to each affected software module. The Certification team applies the changes and resumes the certification process with the new version of the software. The Certification team must maintain a library of the source code and an executable version of the software that they use for testing. Pre-Release Software files are electronic files that can be maintained using any applicable software library system. More detailed descriptions of the Pre-Release Software appear in Sections 5.0 and 5.7.

T8 Failure Reports and Engineering Changes

Each failure encountered by the Certification team as they measure the reliability of the software is recorded in a failure report. The failure report contains the following information: Test Scenario Identification, Description of Test Scenario, Type of Failure, Location of Failure in Scenario, Test Results Prior To Failure, Description of Observed Failure, Specification Reference For Observed Failure, Observations that may help the developers locate the failure source, and Attachments as required. When the failure is resolved the identification of the Engineering Change Notice (ECN) that is intended to fix the failure is noted. The Engineering Change Notice contains the following information: Identification of Failure Report(s) Fixed By This ECN, Textual Description of Change, Routines Modified By ECN, Definition Of Changes Made To Each Routine, Source of Error (Specifications, Black Box, State Box, Clear Box, Code, Previous Modification), and Statement that all required files have been adjusted to reflect change. More detailed descriptions of Failure Reports and Engineering Changes appear in Sections 5.0 and 5.8.

In summary the **state data** is:

T1:	Project Documentation File
T2:	Software Specification Files
T3:	Software Development Files
T4:	Software Certification Files
T5:	Project Management Files
T6:	Unresolved Questions or Issues
T7:	Pre-Release Software
T8:	Failure Reports and Engineering Changes

State Migration and Maintenance

A critical issue in designing a software or system process is to determine where state data will be maintained. There is a struggle between two seemingly contradictory goals: to migrate state data to the lowest level where all the processes not using it cannot access it, and to migrate state data to the highest level where all the processes using it can access it.

In defining the Cleanroom process, it was determined that the best place to keep the state data is at the highest level of the system. This decision was made since nearly all of the high level processes use nearly all of the state data, making modularization illogical. Of course, a different configuration of state data could be designed; but this approach, it was felt, made the description of Cleanroom the clearest and most practical.

3.7 First Level Cleanroom Engineering State Box

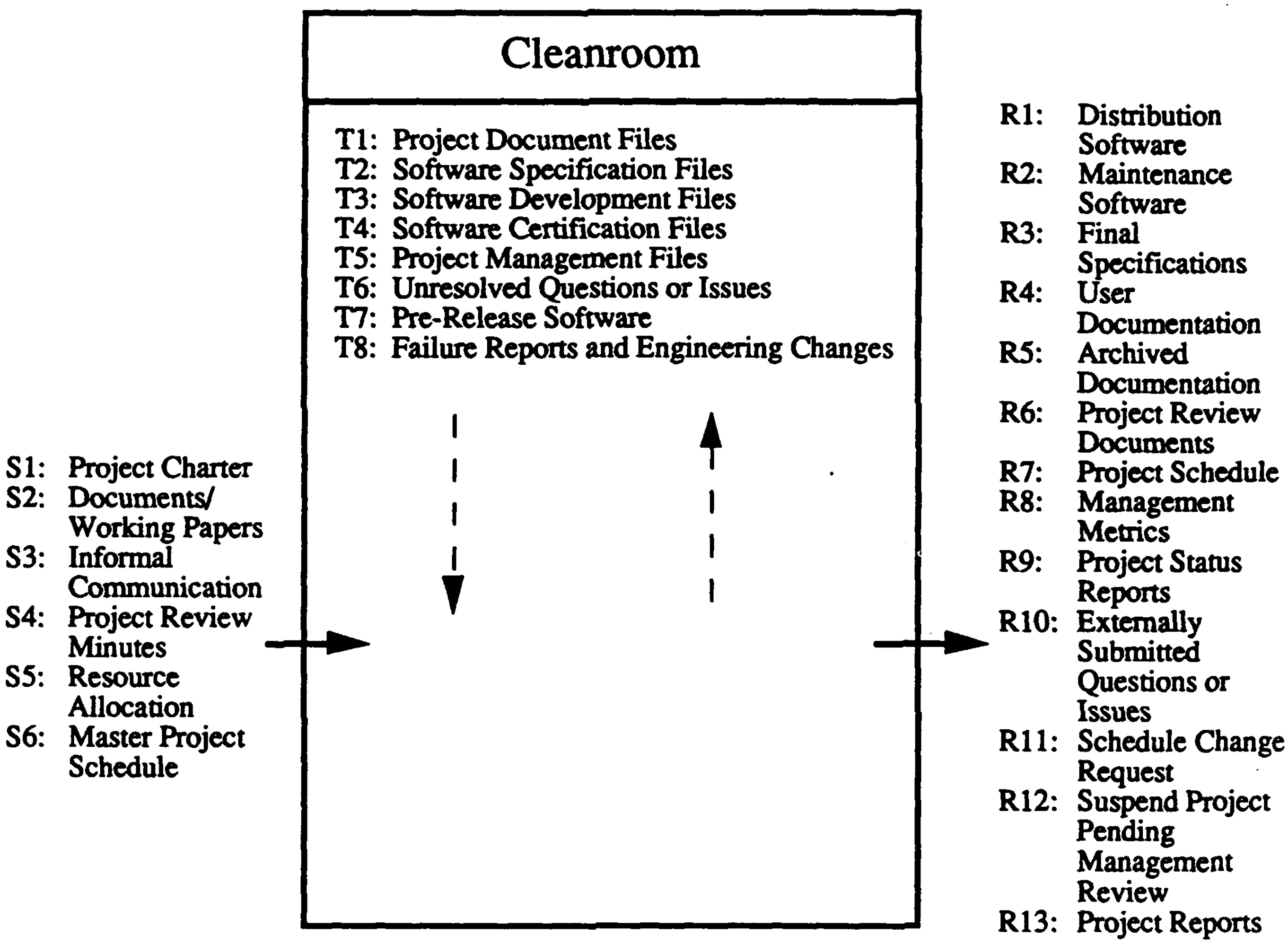
State Box Transformation

The **transformation** for the top level Cleanroom Engineering State Box is a function that maps the **state** and the **current stimuli** to the **responses** and **modified state** at the current level of development. Ideally, this **transformation** should be automatic, but most likely for some period of time, it will take manual writing effort to prepare publishable **responses** from the **state**. The first level Software Engineering State Box is illustrated in Figure 3.7.1. The responses are produced by iteratively modifying the state, as well as from receiving new stimuli. After a number of iterations, transformations and refinements, parts of the state data are presented as responses.

3.8 First Level Cleanroom Engineering Clear Box

The top level **Clear Box** is a control structure that defines the life cycle as all work for translating the **stimuli** into **responses**, and the maintenance of the **state** has been transferred to lower level boxes. Six such lower level Black Boxes have been defined. To indicate that the life cycle is much more complex than a waterfall model, parallel processes are indicated. It is critical to note the relationship between processes using this model. Rather than having the need for each process to communicate with every other process, the state data is updated by each process that uses a particular part of the state data. This prevents the need for communication between processes as the source of information is constantly modified and kept up to date.

Figure 3.7.1 State Box for Cleanroom


Cleanroom
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
S1: Project Charter
S2: Documents/ Working Papers
S3: Informal Communication
S4: Project Review Minutes
S5: Resource Allocation
S6: Master Project Schedule
R1: Distribution Software
R2: Maintenance Software
R3: Final Specifications
R4: User Documentation
R5: Archived Documentation
R6: Project Review Documents
R7: Project Schedule
R8: Management Metrics
R9: Project Status Reports
R10: Externally Submitted Questions or Issues
R11: Schedule Change Request
R12: Suspend Project Pending Management Review
R13: Project Reports

The Cleanroom process is described by the following algorithm.

Clear Box Cleanroom (E0)

Stimuli

S1:	Project Charter
S2:	Documents/Working Papers
S3:	Informal Communication
S4:	Project Review Minutes
S5:	Resource Allocation
S6:	Master Project Schedule

Responses

R1: Distribution Software
R2: Maintenance Software
R3: Final Specification
R4: User Documentation
R5: Archived Documentation
R6: Project Review Documentation
R7: Project Schedule
R8: Management Metrics
R9: Project Status Reports
R10: Externally Submitted Questions or Issues
R11: Schedule Change Request
R12: Suspend Project Pending Management Decision
R13: Project Reports

State

T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes

```
begin

    use E1: Project Invocation
      con
        use E2: Program Management
        use E3: Project Information Management
        do
          do
            use E4: Software Solution Specification
            use E5: Software Development and Certification
          until
            C1: Software Ready for Release?
          od
          use E6: Prepare Final Project Releases
        od
      noc

end Clear Box Cleanroom
```

The top level Clear Box breaks the Cleanroom process into a number of conditions and processes, which represent a high-level view of the parts of the Cleanroom process. General descriptions of each condition and process are given in the following paragraphs, while more detailed descriptions of these and the lower level processes (along with more detailed condition descriptions) are presented in the remainder of Section 3 and in Section 4.

C1 Software Ready for Release?

This condition is the opportunity for management to decide whether the project is ready for release. If the condition is true, the final project releases are prepared; otherwise, the software specifications are rewritten to account for the shortcomings and additional development and certification will commence.

E1 Project Invocation

In this process, the Project Charter, Resource Allocation and Master Project Schedule are received as stimuli, with Externally Submitted Questions and Issues and E1's process completion as responses. The transition leads to staff and resource allocation, with planning the software development approach, and preparation of the initial project schedule.

E2 Program Management

This process receives the completion of process E1 as a stimulus and has Project Review Documents, Project Schedule, Project Status Reports and a Schedule Change Request as responses. The transition for this process includes maintenance of the project schedule, preparing for and conducting project reviews and preparing and submitting status reports.

E3 Project Information Management

Stimuli to this process are the completion of process E1, the Project Charter, Documents/Working Papers, Informal Communications, Project Review Minutes, Resource Allocations and the Master Project Schedule. The responses from this process are Externally Submitted Questions and Issues. Actions done in this process include receiving stimuli and submitting questions or issues.

E4 Software Solution Specification

In this process, the stimuli are the completion of process E1 and the value of condition C1, if it is evaluated. Responses include the completion of process E4, Suspend Project, and Project Reports. Transitions include understanding the problem and solution domains, and writing the software specifications. Additionally, the construction plan is prepared.

E5 Software Development and Certification

This process receives E4's completion as a stimulus and has E5's completion and Project Reports as responses. The actions performed in this process include the development and certification of increments.

E6 Prepare Final Project Releases

The stimulus to this process is the completion of E5, and the response from this process consists of Distribution Software, Maintenance Software, the Final Specifications, User Documentation, Archived Documentation, and Management Metrics. In this process, the final deliverables for the project are packaged and delivered.

A diagram presenting the Cleanroom Engineering Clear Box is presented in Figures 3.8.1 and 4.0.1. {Note to the reader: The graphical descriptions of clear boxes are referenced in both this section and Section 4. Except for the Cleanroom clear box which is duplicated in both sections all other graphical descriptions are only included in Section 4 since that is the section that is designed for long-term reference during projects. Figure 3.9.5.1 includes all the processes imbedded in higher level processes. Some readers may find this figure to be helpful as the develop an understanding of the model.} In the graphical description of Clear Boxes, arrows that divide and lead to multiple process boxes or conditions imply that these processes can work concurrently. That also means that the flow of control can and will be in more than one place at a time. The reasons for this are twofold: 1) the fact that there are three different teams (Specification, Development and Certification) gives each team a different set of responsibilities; and 2) the fact that each team has more than one member makes it possible that each team can be working on more than one process at a time. There is some information on this diagram that is not contained in the algorithm. The main information the diagram shows is which Cleanroom team is responsible for which process: M for Management, S for Specification, D for Development, and C for Certification.

3.9 Second Level Cleanroom Boxes

Each of the six first level Cleanroom processes will be described as a second level box. Four of the six first level Cleanroom processes describe a number of lower level processes which are of sufficient complexity to warrant a second level box being presented here. The boxes to be described on a lower level are the following: Program Management, Project Information Management, Software Solution Specification, and Software Development and Certification. All of these second level processes have lower level Black, State and Clear Boxes.

Since the state data for higher level processes are available to lower level dependent processes, it has been chosen for clarity to repeat all first level state data in the second level boxes with the label Outer State

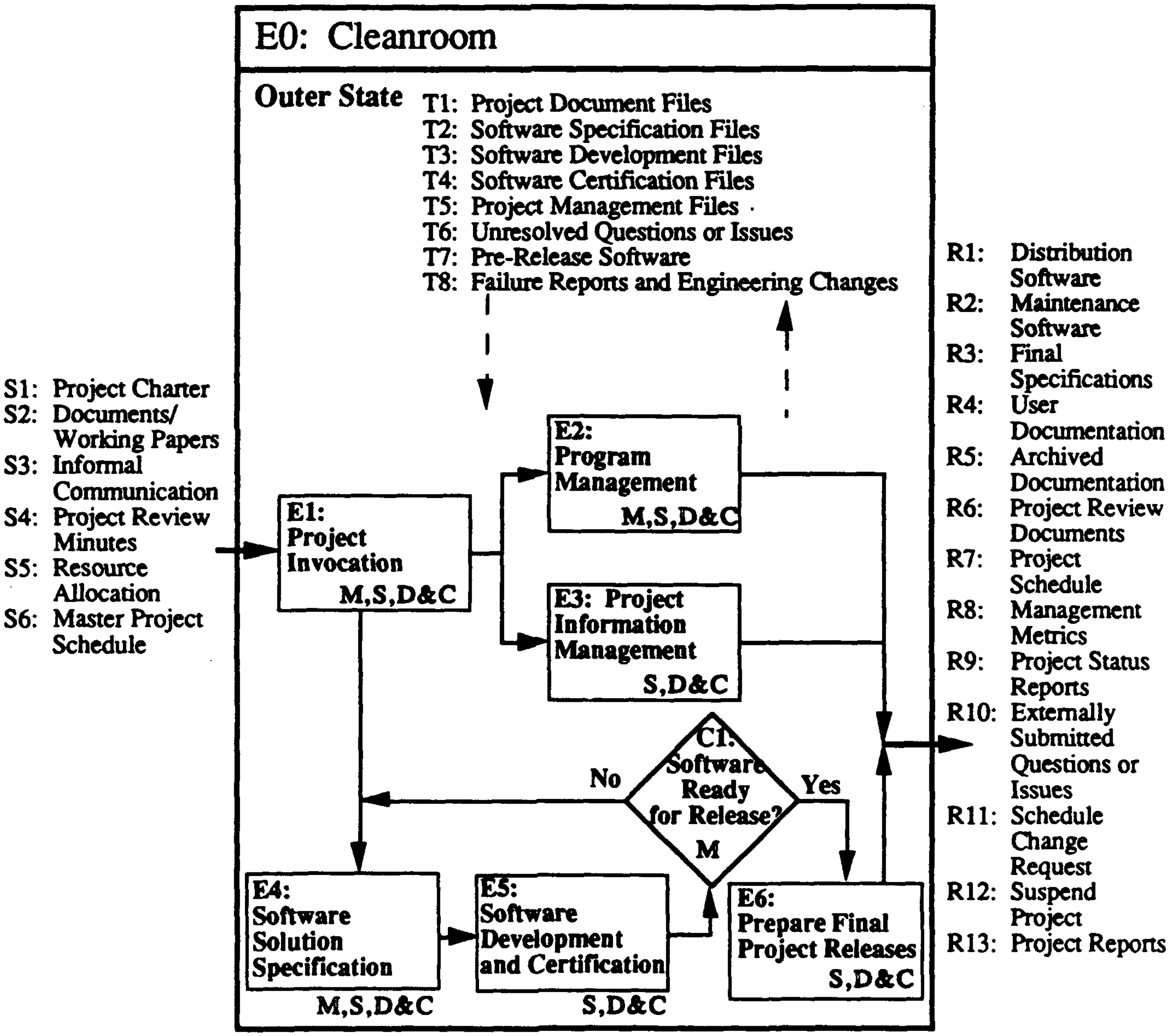
3.9.1 Program Management

The Black Box for Program Management (process E2) has only one stimulus. It is a signal which states that process E1 has been completed. This is an internal stimulus, since it did not appear in the first level Black Box for Cleanroom. Additionally, Program Management has four responses: R7 - the Project Schedule, R6 - Project Review Documentation, R9 - Project Status Reports, and R11 - a Schedule Change Request. The transformation for E2 includes the maintenance and release of the project schedule, preparing and conducting project reviews and preparing and submitting status reports.

The State Box for Program Management is the same as the Black Box. This is the case because no lower level state data requirements were found for the Program Management process.

The Clear Box for Program Management, where actual processing is defined for the process, is a refinement of the black and State Boxes.

Figure 3.8.1 Clear Box for Cleanroom


E0: Cleanroom
Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
S1: Project Charter
S2: Documents/ Working Papers
S3: Informal Communication
S4: Project Review Minutes
S5: Resource Allocation
S6: Master Project Schedule
E1: Project Invocation M,S,D&C
E2: Program Management M,S,D&C
E3: Project Information Management S,D&C
C1: Software Ready for Release? M
No
Yes
E4: Software Solution Specification M,S,D&C
E5: Software Development and Certification S,D&C
E6: Prepare Final Project Releases S,D&C
R1: Distribution Software
R2: Maintenance Software
R3: Final Specifications
R4: User Documentation
R5: Archived Documentation
R6: Project Review Documents
R7: Project Schedule
R8: Management Metrics
R9: Project Status Reports
R10: Externally Submitted Questions or Issues
R11: Schedule Change Request
R12: Suspend Project
R13: Project Reports

The Program Management process is described by the following algorithm.

Clear Box Program Management (E2)

Stimuli

I1: E1 Completed

Responses

R6: Project Review Documentation
R7: Project Schedule
R9: Project Status Reports
R11: Schedule Change Request

Outer State

T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes

```
begin

    do
        con
            if C2: Schedule to be Modified/Published?
                then
                    use E7: Maintain Project Schedule
            fi
            if C3: Work Complete/Review Scheduled/Major Problem?
                then
                    use E8: Prepare for & Conduct Project Review
            fi
            if C4: Status Scheduled?
                then
                    use E9: Prepare & Submit Status Reports
            fi
        noc
    until
        Completion Conditions achieved
    od

end Clear Box Program Management
```

The second level Clear Box breaks the Program Management process into a number of conditions and processes, which represent a low-level view of the Program Management process. General descriptions of each description and process are given in the following paragraphs, while a more detailed description of these and the lower level processes (along with detailed condition descriptions) are provided in Section 4.

C2 Schedule to by Modified/Published?

This condition is true when there is a reason for the schedule to be modified. The reasons may include, among other things, a new Master Project Schedule or the need for new tasks to be assigned.

C3 Work Complete/Review Scheduled/Major Problem?

This condition is true when the need for a review arises. This "need" may be scheduled or unscheduled.

C4 Status Scheduled?

This condition is true when the project schedule dictates that a status report is due to be submitted, or measures need to be accumulated.

E7 Maintain Project Schedule

In this process, the completion of process E1 is the stimulus and the Project Schedule and Schedule Change Request are the responses. That transition consists of updating the project schedule and completing task assignments.

E8 Prepare For & Conduct Project Review

This process receives the completion of process E1 as a stimulus and returns the Project Review Documentation as a response. The transition function is fairly obvious from the process name.

E9 Prepare & Submit Status Reports

The stimulus to this process is also the completion of process E1, with Project Status Reports being the responses. The transition function includes the gathering of measurement data, as well as the preparation and submission of status reports.

Graphically, the Program Management process appears in Figure 4.2.1.

3.9.2 Project Information Management

The Black Box for Project Information Management (process E3) has six external stimuli and one internal stimulus. The external stimuli are: the Project Charter (S1), Documents/Working Papers (S2), Informal Communication (S3), Project Review Minutes (S4), Resource Allocation (S5), and the Master Project Schedule (S6). The internal stimulus is a signal which states that process E1 is completed. Project Information Management also has one external response: R10, which is an externally submitted question or issue. There are also a number of transformations to the first level (outer box) state, which are not explicitly stated here. The transformation for E3 includes receiving and filing all externally received stimuli. The stimuli are filed in a manner which makes the information easily accessible by the entire staff. Additionally, questions or issues which are to be submitted or resolved internally or submitted externally are organized in this process, since this is where the responses to those questions are received.

The State Box for the Project Information Management is the same as the Black Box. This is the case because no lower level state data requirements were found for the Project Information Management process.

The Clear Box for Project Information Management, where actual processing is defined for the process is a refinement of the black and State Boxes.

The Project Information Management process is described in the following algorithm.

Clear Box Project Information Management (E3)

Stimuli

I1:	E1 Completed
S1:	Project Charter
S2:	Documents/Working Papers
S3:	Informal Communication
S4:	Project Review Minutes
S5:	Resource Allocation
S6:	Master Project Schedule

Responses

R10:	Externally Submitted Question or Issue

Outer State

T1:	Project Document Files
T2:	Software Specification Files
T3:	Software Development Files
T4:	Software Certification Files
T5:	Project Management Files
T6:	Unresolved Questions or Issues
T7:	Pre-Release Software
T8:	Failure Reports and Engineering Changes

```
begin

    do
        con
            if C5:  Stimuli Received?
                then
                    use E10:  Receive Stimuli
            fi
            if  C6:  Questions or Issues?
                then
                    use E11:  Submit a Question or Issue
            fi
        noc
    od

end Clear Box Project Information Management
```

The second level Clear Box breaks the Project Information Management process into a number of conditions and processes, which represent a low-level view of the parts of the Project Information Management process. General descriptions of each condition and process are given in the following paragraphs, while a more detailed description of these and the lower level processes (along with more detailed descriptions of conditions) are provided in Section 4.

C5 Stimuli Received?

This condition is true whenever an external stimuli is made available to the engineering team staff.

C6 Questions or Issues?

This condition is true whenever there is a question or an issue that cannot be resolved by a member of the engineering team. In this case, the questions or issues are submitted to others within the engineering team, or externally to others outside of the engineering team for resolution.

E10 Receive Stimuli

The process receives the completion of process E1, the Project Charter, Documents/Working Papers, Informal Communication, Project Review Minutes, Resource Allocation, and the Master Project Schedule as stimuli, and returns the completion of E10 as a response. This process merely receives and stores the external stimuli.

E11 Submit a Question or Issue

The stimulus to this process is the completion of process E1, responding with Externally Submitted Questions or Issues. The transition is identical to the process name.

Graphically, the Project Information Management process appears in Figure 4.3.1.

3.9.3 Software Solution Specification

The Black Box for Software Solution Specification (process E4) has two internal stimuli. One is a signal which states that process E1 has been completed, and the other is a flag which holds the result of condition C1. C1 is activated only when this is not the first time the Software Solution Specification process is being invoked. Additionally, Software Solution Specification has three responses: a signal that the process has been completed, R12 which is a decision to suspend the project, and R13 which are project reports (including working papers from the state data, where the working papers are converted into project reports). The transformation for E4 includes understanding the problem and solution domains, and writing the software specifications. Additionally, the construction plan can be written or rewritten, depending on how the process was invoked.

The State Box for Software Solution Specification is the same as the Black Box. This is the case because no lower level state data requirements were found for the Software Solution Specification process.

The Clear Box for Software Solution Specification, where actual processing is defined for the process, is a refinement of the Black and State Boxes.

The Software Solution Specification process is described in the following algorithm.

Clear Box Software Solution Specification (E4)

Stimuli

I1: E1 Completed
I2: C1 Software Not Ready for Release

Responses

I2: E4 Completed
R12: Suspend Project Pending Management Decision
R13: Project Reports

Outer State

T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes

```
begin
    do
        if (C1 not a stimulus)
            then
                con
                    use E12: Understand Problem Domain
                    use E13: Understand Solution Domain
                    use E14: Write Specifications
                noc
                use E15: Write Construction Plan
            else
                if C7: Continue with Project?
                    then
                        con
                            use E12: Understand Problem Domain
                            use E13: Understand Solution Domain
                            use E14: Write Specifications
                        noc
                        use E15: Write Construction Plan
                    else
                        Suspend project (R12)
                        Entire engineering team staff will archive state data for future use
                        Exit process
                fi
        fi
    od
end Clear Box Software Solution Specification
```

The second level Clear Box breaks the Software Solution Specification process into a number of conditions and processes, which represent a low-level view of the parts of the Software Solution Specification process. General descriptions of each condition and process are given in the following paragraphs, while a more detailed description of these and the lower level processes (and more detailed condition descriptions) are provided in Section 4.

C7 Review to Decide - Continue with Project?

This condition occurs when the Software Solution Specification process occurs more than once for a project. It will evaluate to true whenever the manager determines that the project can be salvaged by developing a new Construction Plan. It evaluates to false whenever the manager determines that the project cannot be salvaged as is and should become the decision of management as to what the next step is.

E12 Understand Problem Domain

In this process, the stimulus is the completion of process E1 and the response is the completion of process E12. The transition is defined by the process name.

E13 Understand Solution Domain

With this process, the completion of process E1 is the stimulus again, with the completion of process E13 being the response. This transition is evident from the process name.

E14 Write Specifications

This process receives the completion of process E1 as a stimulus and returns the completion of process E14 as the response. In this process, the Specification team writes the first five volumes of the six-volume specifications document. These volumes are: the Mission Volume, the User's Reference Manual Volume, the Black Box Function Volume, the Black Box Function Verification Volume, and the Usage Profile Volume.

E15 Write Construction Plan

The stimuli to this process are the completion of process E12, E13 and E14. The response is the completion of process E15. The transition is simply the preparation of the final volume of the specifications (the Construction Plan).

Graphically, the Software Solution Specification process appears in Figure 4.4.1.

3.9.4 Software Development and Certification

The Black Box for Software Development and Certification (process E5) has one internal stimulus. It is a signal which states that process E4 has been completed. Additionally, Software Development and Certification has two responses, one external and one internal: a signal (I3) that the process has been completed, and R13 which are any project reports (which includes working papers from the state data, where the working papers are converted into project reports). The transformation for E5 includes the development and certification of the software increments.

The State Box for Software Development and Certification is the same as the Black Box. This is the case because no lower level state data requirements were found for the Software Development and Certification process. The Clear Box for Software Development and Certification, where actual processing is defined for the process, is a refinement of the Black and State Boxes.

The Software Development and Certification process is described in the following algorithm.

Clear Box Software Development and Certification (E5)

Stimuli

I3:	E4 Completed

Responses

I4:	E5 Completed
R13:	Project Reports

```
Outer State
      T1:    Project Document Files
      T2:    Software Specification Files
      T3:    Software Development Files
      T4:    Software Certification Files
      T5:    Project Management Files
      T6:    Unresolved Questions or Issues
      T7:    Pre-Release Software
      T8:    Failure Reports and Engineering Changes

begin

      do
         con
            use E16: Increment Development (for increment 1)
            use E16: Increment Development (for increment 2)
            ...
            use E16: Increment Development (for increment i)
            ...
            use E16: Increment Development (for increment n)
            while C8: Next Increment in Construction Plan Ready?
               do
                  use E17: Increment Certification
                  if C9: Final Increment?
                     then
                        Exit process
                  fi
               od
         noc
      until
         Completion Conditions achieved
      od

end Clear Box Software Development and Certification
```

The second level Clear Box breaks the Software Development and Certification process into a number of conditions and processes, which represent a low-level view of the parts of the Software Development and Certification process. General descriptions of each condition and process are given in the following paragraphs, while a more detailed description of these and the lower level processes (and more detailed condition descriptions) are provided in Section 4.

C8 Next Increment in Construction Plan Ready?

This condition is true whenever the Development team has next increment (according to the order dictated in the Construction Plan) ready for delivery to the Certification team.

C9 Final Increment?

This condition is true whenever the final increment has been integrated into the software system and has completed certification. Otherwise, development and certification will continue.

E16 Increment Development

This process receives the completion of process E4 as stimulus, and returns the code for an increment as the response. The transition consists of developing an increment (E18), preparing certification tests for an accumulated increment (E19), updating the specifications (E20), and increasing the understanding of the problem and solution domains as required (E21). These processes are described in detail in Section 4.

E17 Increment Certification

The stimulus to this process is the code for an increment, with the response being the completion of process E17. Increasing the understanding of the problem and solution domains (E21), building the system with the new increment (E22), certifying the accumulated increment (E23), and correcting the code for the accumulated increment (E24) compose the transitions for the process. These processes are described in greater detail in Section 4.0.

Graphically, the Software Development and Certification process appears in Figure 4.5.1. Detailed descriptions of all 25 processes appear in Section 4.

3.9.5 The Detailed Cleanroom Process

As previously stated, the Cleanroom Engineering process breaks down into a total of 25 processes. All 25 processes have been described in the previous sections, but the relationship between all processes needs to be described. That relationship is illustrated in Figure 3.9.5.1, where E0 is implicit as the process which contains the others. This figure adds no new information, but it has been found to be helpful when preparing a realization of the model for a specific project.

3.10 Project Reviews

Project reviews are an integral part of software development projects and are handled by the model no matter when they are scheduled. There are many alternative review schedules. The precise review schedule for a project depends on many things such as the wishes of the project sponsor, the custom of the organization, etc. The purpose of this section is to relate project reviews to the Cleanroom Engineering project cycle and suggest a schedule of reviews that relates well to Cleanroom Engineering.

3.10.1 The Purpose Of Project Reviews

Project reviews are held so the project sponsor can appraise progress and determine if the project is still heading in the desired direction. As a result of the review, the project sponsor can take actions as follows:

1. Determine that the project is in serious trouble because it is heading in an undesired direction. In this case, the project sponsor may suspend the project or suggest a significant mid-course correction.

State Data

T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software.Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes

S1: Project Charter
S2: Documents/ Working Papers
S3: Informal Communication
S4: Project Review Minutes
S5: Resource Allocation
S6: Master Project Schedule

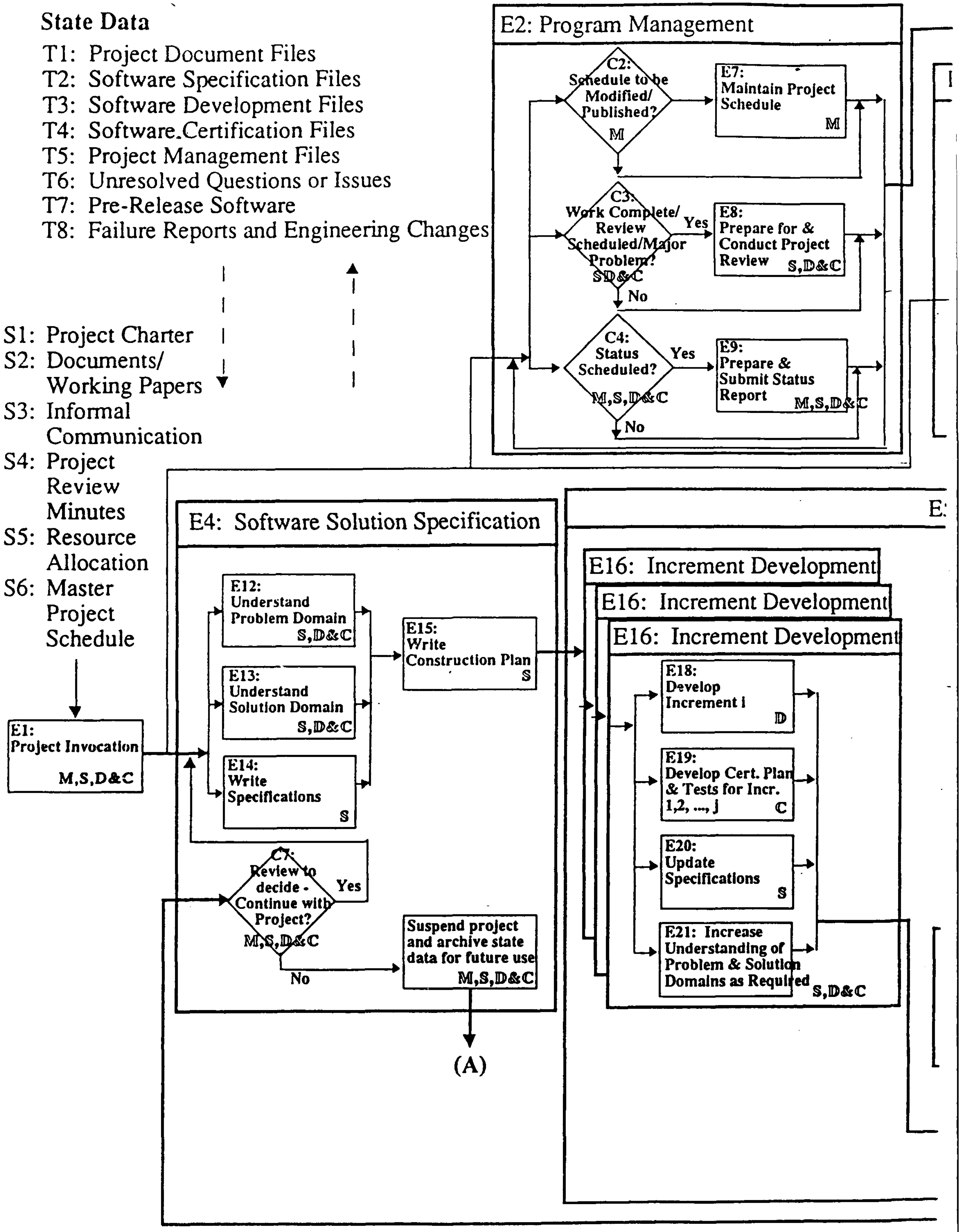
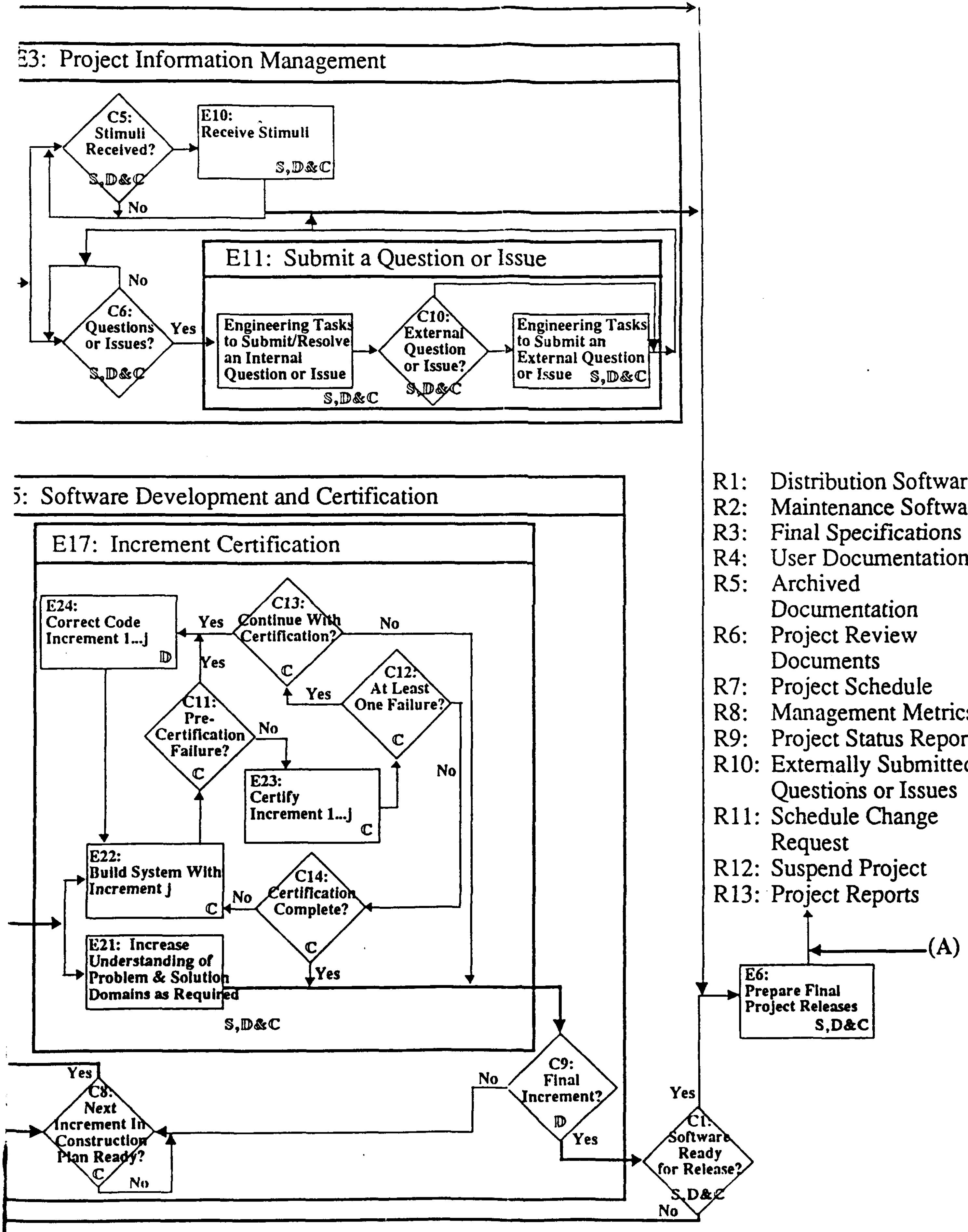

E2: Program Management
C2: Schedule to be Modified/ Published? M
E7: Maintain Project Schedule M
C3: Work Complete/ Review Scheduled/Major Problem? S,D&C
Yes
No
E8: Prepare for & Conduct Project Review S,D&C
C4: Status Scheduled? M,S,D&C
Yes
No
E9: Prepare & Submit Status Report M,S,D&C
E1: Project Invocation M,S,D&C
E4: Software Solution Specification
E12: Understand Problem Domain S,D&C
E13: Understand Solution Domain S,D&C
E14: Write Specifications S
E15: Write Construction Plan S
C7: Review to decide - Continue with Project? M,S,D&C
Yes
No
Suspend project and archive state data for future use M,S,D&C
(A)
E:
E16: Increment Development
E16: Increment Development
E16: Increment Development
E18: Develop Increment i D
E19: Develop Cert. Plan & Tests for Incr. 1,2, ..., j C
E20: Update Specifications S
E21: Increase Understanding of Problem & Solution Domains as Required S,D&C

Figure 3.9.5.1


E3: Project Information Management
C5: Stimuli Received?
S,D&C
No
E10: Receive Stimuli
S,D&C
No
C6: Questions or Issues?
S,D&C
Yes
E11: Submit a Question or Issue
Engineering Tasks to Submit/Resolve an Internal Question or Issue
S,D&C
C10: External Question or Issue?
S,D&C
Engineering Tasks to Submit an External Question or Issue
S,D&C
5: Software Development and Certification
E17: Increment Certification
E24: Correct Code Increment 1...j
D
Yes
C13: Continue With Certification?
C
No
Yes
C11: Pre-Certification Failure?
C
No
Yes
C12: At Least One Failure?
C
No
E23: Certify Increment 1...j
C
E22: Build System With Increment j
C
No
C14: Certification Complete?
C
Yes
E21: Increase Understanding of Problem & Solution Domains as Required
S,D&C
Yes
C8: Next Increment In Construction Plan Ready?
C
No
No
C9: Final Increment?
D
Yes
Yes
C1: Software Ready for Release?
S,D&C
No
E6: Prepare Final Project Releases
S,D&C
(A)

R1: Distribution Software
R2: Maintenance Software
R3: Final Specifications
R4: User Documentation
R5: Archived Documentation
R6: Project Review Documents
R7: Project Schedule
R8: Management Metrics
R9: Project Status Reports
R10: Externally Submitted Questions or Issues
R11: Schedule Change Request
R12: Suspend Project
R13: Project Reports

(This page intentionally left blank)

2. Determine that the project is in serious trouble because the project development practices being used are inadequate; or if they are adequate, they are being followed so poorly that the resulting work will be substandard. In either case, the project sponsor may suspend the project or suggest significant corrective action.

3. Determine that the project is proceeding in the desired direction, and the project performance practices are adequate and are being followed so the work product will be of the expected quality. In this case, the project sponsor is likely to continue with the project offering suggestions and/ or instructions to the project team to correct any deficiencies found in the product as a result of the review.

Software project reviews typically focus on all aspects of the product from the highest to lowest detail since project sponsors place very low reliance in contemporary heuristic, trial-and-error software development practices. In other disciplines, reviews focus on the external aspects of the product and, in addition, take a sample of the internal details to determine if proper project performance practices are being followed. If the sample indicates sound engineering practices are being followed, the sponsor has confidence that the entire project will meet its quality goals.

3.10.2 Project Reviews and Cleanroom Engineering

The model of software development practices presented in this document is compatible with any review schedule the project sponsor desires. A review schedule that corresponds to critical points in the project cycle is more productive than a review schedule that is organized to the calendar and thus misses critical project points. When a software development project is organized as suggested by this model, any schedule is feasible and will cause minimal impact on project performance.

It is envisioned that the development agreement will specify the reviews and the content of the reviews. In this way both the sponsor and the developer know what to expect and how to prepare. There are three stages to a review: the preparatory stage, the review itself and the review follow-up. In this model, the following acts are envisioned in each of the three stages.

Preparing For A Review. Typically, the review plan as contained in the development agreement will specify the items (and very often the format for the items) that the reviewers want to examine in each review. In this model of software development, it is envisioned that review documents will be prepared directly from state data files. Review documents should not contain any data or information that is not in the state data. In fact, the review documents should be prepared from the state data by anyone knowledgeable in software engineering practices. An alternative that eliminates the need to prepare special review documents is to give the reviewers direct, read only, access to the state data files. In any event, all the material required for the review team is prepared and given to the review team in advance for study. Typically, at the review the reviewers will want an introductory presentation to guide them through the materials and the current state of the product. This presentation will not be part of the state data prior to the review, but it must be prepared from the state data materials.

Conducting The Review. The reviewers will be in control. The desired presentation should be delivered. The reviewers should be given the time they desire to read and study material before the review. All their questions during the review should be answered as clearly as possible using material contained in the state data. Reviewers will typically focus on several issues by trying to answer two classes of questions:

1. Will the product as designed provide satisfaction to its users assuming it is implemented in accordance with the design? Is it possible to design a product that will provide more satisfaction while meeting other economic constraints?

2. Will the engineering practices as defined permit engineers to construct a product that is in accordance with its design? Are the defined engineering practices being followed?

The first class of questions is answered by examining the specification and the verification argument contained in the specification. The verification argument should be evaluated to determine if it adequately shows that if software existed that meets the specifications, the software will complete the mission defined for the software.

The second class of questions is answered by examining the detailed engineering practices and then conducting a random sample to determine how well the practices are being followed. In the case of Cleanroom, the best things to sample are the verification arguments and how the engineering teams are performing against the completion conditions defined for each process. Completion conditions are introduced in detail in Section 4.0. If the review team is satisfied that the engineering team is only proceeding from one process to the next when the completion conditions are satisfied and that all verification arguments are correct, the review team can then conclude the Cleanroom processes are being followed. The other element review teams should carefully examine are the results of software certification tests. These tests are measurements of the underlying process. If the software certification tests show few failures and indicate a high MTTF, the review team can conclude that the sample of tests conducted indicates high quality. Since these tests were drawn at random from the universe of all possible tests and they indicate a high MTTF, the conclusion is that the development process is in a state of intellectual control. As a result it can be concluded that the software, as it is being produced, will exhibit similar quality for its full range of operations.

The Review Follow Up. The review team puts its conclusions in writing in a formal report. The report includes recommendations in the form of action items. The recommendations take two forms. First, recommendations about the process. Second, recommendations about the product. The sponsor will accept some recommendations and reject others. These accepted recommendations are modifications to the project charter which from that point forward must be included in the product or they are recommendations to modify the process which must be accounted for as part of the completion conditions for the appropriate process.

3.10.3 A Feasible Review Schedule For Cleanroom Projects

There are many feasible review schedules for Cleanroom projects. The actual project review schedule will be specified by agreement between the project sponsor and the project developer. The following review schedule is a useful review schedule since it fits into the Cleanroom project cycle. In this recommendation, there are four normal reviews and one exceptional review. The reviews are:

Software Requirements Review. This review should be held as soon as possible after the software project is initiated to determine if there is sufficient understanding of the mission the software is to perform. The software team must participate in this review so they can agree that sufficient understanding does exist for them to proceed. A comment draft of the mission volume of the specifications should be available.

Preliminary Design Review. This review should be held at the completion of the Software Solution Specification Process (E4) or Write Specifications process (E14) to insure that if the software is developed as specified, it will be able to fulfill the mission that has been assigned to the software.

Critical Design Review. This review should be held at the completion of the Increment Development Process (E16) for each increment to insure the software as designed meets the specifications and that the certification tests are adequate.

Software Acceptance Review. This review should be held at the completion of certification of the last increment prior to Preparing Final Project Releases (E6) to insure the sponsor is satisfied with the software as developed.

Increment Failure Review. This review should be held whenever the Certification team determines that an increment as implemented is too failure-laden to continue certification. The purpose of this review is to investigate the failure and recommend corrective action.

3.11 Model Comparison

The purpose of this section is to answer such questions as: How does the Cleanroom Engineering Project Cycle differ from other project cycles? How is it the same? What are the advantages of this modeling approach over other modeling approaches?

The first point to make is that all models of the software development process are similar to the extent that they all deal with how to manage a complex process where the principal processing components are highly trained professionals. The models all need to deal with overhead functions such as status reports, project reviews, project scheduling, etc. At the working level they all need to deal with specification development, software design and implementation and software certification. All models need to deal with process interaction by specifying the control flow among processes. At the control level all models must deal with determining when a process is complete and when the software is complete. The software is complete where it will perform its assigned mission with satisfactory reliability. All models must deal with how to account for work-in-process as the engineers move from a mission statement for the software to the actual software.

Section 2 illustrates much of what makes the Cleanroom environment different from the environment which is associated with conventional trial-and-error software development practices. The Cleanroom practices are used at the working levels by the engineers who are actually doing the hard work of specifying, developing and certifying. Therefore, the Cleanroom practices are used in the leaf nodes of the process hierarchy. These are the 12 processes that are used by the three control processes: E4: Software Solution Specification, E16: Increment Development and E17: Increment Certification. These three control processes were developed in this section. The Engineering Tasks for performing these 12 processes, which are the 12 processes directly effected by Cleanroom Engineering practices, are defined in Section 4.

Section 4 also defines the engineering tasks for all the other 13 processes. These 13 processes are also affected by Cleanroom. The big advantage Cleanroom Engineering practices offer managers is that they provide them with the facilities required to manage projects since the project is performed in a state of intellectual control. What makes a software project difficult to manage is that the development effort is not under intellectual control. This same condition also makes it very difficult, if not impossible, to develop a simple meaningful model of the software development process. Therefore, with Cleanroom Engineering, since the processes where the hard engineering work is done are under intellectual control, the processes that control the work flow and the provide the services to support the hard engineering work can be specified in a simple-to-understand-and-use form.

What makes this model of the software development process different than other models are the Cleanroom methods. Cleanroom makes it possible to specify clear engineering tasks for the hard engineering processes. Cleanroom makes the management task possible since it eases the modeling requirements for the control flow and management tasks. Cleanroom makes it possible to specify predicates (True/False conditions) to determine when a process is complete, which eases both the management and modeling effort.

Another important feature of the model is the treatment of in-process work as state data. It is, in fact, state data that represents the current state of the understanding and the transformation of the stimuli to the desired responses. The treatment of this information in the model as state data simplifies the model since many data flows are not required. It is not necessary to invent internal products solely for the purpose of accounting for work or to pass data. Treating in-process work as state data on a real project has these same advantages since it models how engineers really want to work. In addition, by treating in-process work as state data, the in-process work is moved into public view where it belongs. This is in contrast to the current situation where many in-process work is the private preserve of individual engineers. Another advantage of this relationship is that it does not require the engineers to keep two sets of books - one that they use to get the information they need, and one that contains the defined intermediate products required to satisfy project monitors and get paid.

Below, each of the 25 processes are enumerated along with comments about how they relate to other modeling efforts.

E0: Cleanroom
Defines the top level control process so process differences are largely invisible. The differences that are visible include: (1) enumeration of all stimuli and responses, (2) the central role of the state data, and (3) since all results are maintained as state data, the final process is to prepare the final project releases from the state data.

E1: Project Invocation
All models provide this function either explicitly or implicitly.

E2: Program Management and used processes
E7: Maintain Project Schedule
E8: Prepare for and Conduct Project Review
E9: Prepare and Submit Status Report
All models account for these activities in some way.

E3: Project Information Management and used processes
E10: Receive Stimuli
E11: Submit a Question or Issue
Al' models account for these activities in some way.

E4: Software Solution Specification
This is the control process for preparing the specification and the process of suspending the project. What is different about this control structure is the leaf processes which are being controlled and since the leaf nodes are different, the control structure may be different.

E5: Software Development and Certification and used processes
E16: Increment Development
E17: Increment Certification
These are control structures for developing and certifying the software. What is different about this control structure is the leaf processes which are being controlled; and since the leaf nodes are different, the control structure may be different.

E12: Understand Problem Domain
E13: Understand Solution Domain
These two engineering processes contain the engineering activities required to perform the bottom-up thinking that is required to support specification development. The exact nature of the tasks that need to be performed is highly dependent on the project and what is known when the project was initiated. Many models account for this bottom-up thinking in some way and others ignore the difficulty so there is not a clear separation between bottom-up thinking and top-down design. Also models may not make a clear distinction between bottom-up thinking designed to support problem and solution domain understanding.

E14: Write Specifications
All models account for writing specifications. With Cleanroom there is an instance on clear, complete and formal specifications before top-down development is initiated. The format of a Cleanroom specification is different. Therefore, the engineering tasks to produce the specification are different. With Cleanroom, there is an instance that the specifications must be verified, to insure that if the software is developed and certified as meeting the specifications, it will perform the mission assigned to the software. This is because Cleanroom clearly separates the easier problem of developing software from the more difficult problem of inventing specifications. This separation is required so that the software, which is a rule, can be verified and certified to the function which is defined in the specifications.

E15: Write Construction Plan
All models somehow prepare a plan for constructing software in increments. The Cleanroom requirement for user executability for each accumulating set of increments is a necessary condition to utilize usage testing at each certification step. This requirement causes this process to be different.

E18: Develop Increment i
All models account for software development in one or more processes. The Cleanroom model is much simpler in this aspect since the Cleanroom technologies enable engineers to develop and verify software under intellectual control. Therefore, the engineering tasks used to perform this process reflect the use of these technologies.

E19: Develop Certification Tests for Increments 1...j
Cleanroom relies on usage testing to certify software. Most conventional software development practices and hence models rely on some form of coverage testing. Thus the engineering tasks that define this process are different.

E20: Update Specifications
All models in some way account for the need to maintain specifications current. Cleanroom requires that the Specifications always be current and that the specifications be decomposed to support incremental construction, so this process is difficult.

E21: Increase Understanding of Problem and Solution Domains as Required
All model account for continuing bottom-up exploration in some way. Continued bottom-up exploration is vital to permit developers to explore ahead of the design to help make current design trades.

E22: Build System With Increment j
This process is different since conventional practice assumes systems are built as a by-product of the development effort. With Cleanroom there is clear separation between development ar:d certification.

E23: Certify Increment 1...j
This process is different since in Cleanroom the software is built in a series of accumulating increments, all under statistical quality control. This process is where the statistical quality control takes place.

E24: Correct Code for Increments 1...j
This process is different with Cleanroom since failures are observed by the Certification team and formally passed back to the Development team for correction. This converts failure correction from a "private backroom" process into a "public process" with high visibility. In addition, when the Development team makes a correction, it must trace the failure to its source and correct the entire design. In some cases, this means going back to the Box Structure design. This requirement reduces the possibility that a fix to a failure will result in the condition of local correctness and global incorrectness. With Cleanroom, developers have many less failures to remedy. This high visibility makes it easier for the development organization to lc..rn to prevent future failures.

From the above description, it is apparent that the differences are primarily in the leaf processes where the Cleanroom technologies are used to establish intellectual control over the design process. This, in turn, permits the control processes to become simpler since they are being used to control processes that are understood and are under intellectual control.

From this modeling effort, it is reasonable to draw the fo..owing conclusions.

1. It is possible to fit the Cleanroom technologies into a total software development process.

2. The model of the software development process which includes Cleanroom technologies is simpler than process models that attempt to depict traditional trial-and-error software development practices.

3.12 Using The Process Model

This process model is a description of the Cleanroom process, not a prescription of how to complete the Cleanroom process in all situations. The process model is not used directly to allocate effort to and then control the performance of a project. The process of using the model to plan for, to schedule and then control a project requires four steps as follows:

1. The process model is used to define a list of all the processes required for the project of interest - the project processes. To prepare the list requires knowledge of the specific project. Typical project details include the number of increments, the number of reviews, when they are to occur, etc.

2. The process model is used to determine the precedence relationships between the project processes. The result is a precedence diagram, commonly called a PERT chart. Given project knowledge, it is possible to assign the effort required to perform each of the project processes. Therefore, it is possible to determine the critical path through the precedence diagram. The result is the minimum project duration.

3. Then, given resources to perform the project, it is possible to allocate the project processes to the resources and develop a schedule. There are many possible schedules that can be used to perform a project. Project managers try to find the best schedule. "Best" is defined according to many factors including risk minimization.

4. Then, given a schedule, the project manager makes real-time observations and, as a result, modifies the schedule to account for difficulties or to take advantage of opportunities. Examples include:

   A process is scheduled to begin and, for any number of reasons, it is determined the process should not start as scheduled. As a result, the project manager does not issue the instructions to invoke the process.

   A process is in progress and either because of events observed in the performance of that process or in some other process the manager decides to interrupt performing that task.

Section 4
Process Basis for Cleanroom Engineering

Organization

The purpose of this section is to provide a reference base for users of the model. A manager will be able to use this section as a basis for guiding an engineering staff that is developing software using Cleanroom Engineering and/or tailoring the process to meet specific project/organizational requirements. This section is not designed to teach professionals how each Cleanroom engineering task should be performed. A prescriptive model for Cleanroom is dependent on the organization, environment, project, etc. The descriptive model presented in the subsequent pages is also not immutable. It will change and expand, as engineering tasks become clearer and completion conditions become more specific.

The twenty-five processes which make up the Cleanroom Engineering process model (numbered E0 through E24) are defined individually in the subsequent sections. Each process is described in the order it was referred to in Section 3. Each section starts on a new page to make it easier to support the use of the process model on individual projects. For each process, a consistent notation and structure are used and are defined as follows:

Process Summary A general description of the process is presented in this section.

Outer State This section lists state data that is accessible by the process. State data that is in bold print is actually in the specific process. State data in plain text is not used in the specific process, although it may be used in lower level processes of the original process.

Process Illustration The process illustration presents the control flow for the process in a graphical format.

Process Completed By This section details the people responsible for completing this process. The responsible people may be the engineering team manager, Specification team, Development team or Certification team. On the figures, all processes and conditions have the people that complete the process or condition written in outline print.

Previous Process The Cleanroom Engineering process that occurred previous to this process is listed here.

Pre-Conditions Any conditions that appear between this process and the previous one are listed, along with the cases that lead to this process.

Subsequent Process The process that will occur immediately after this process is listed here.

Stimuli The stimuli for a box can be external stimuli, control messages, process messages or parameters (such as code).

Responses The responses from a box can be external responses or process messages or parameters (internal responses).

State Data Usage Use of and modification to state data is listed in this section.

Process Description The steps that one must follow to complete a process are given in this section. These steps are either lower level processes or engineering tasks. Additionally, conditions that appear in the box are also explained in greater detail.

Measurement Data Generated The categories of data for measurement that are generated in this process are listed in this section.

Completion Conditions To exit any process, the completion conditions for the process must be met. All of the completion conditions must be answered affirmatively to exit the process. Otherwise the process cannot be considered complete. Forms containing a listing of the completion conditions are included in Appendix A for easy reproduction and use by engineering teams.

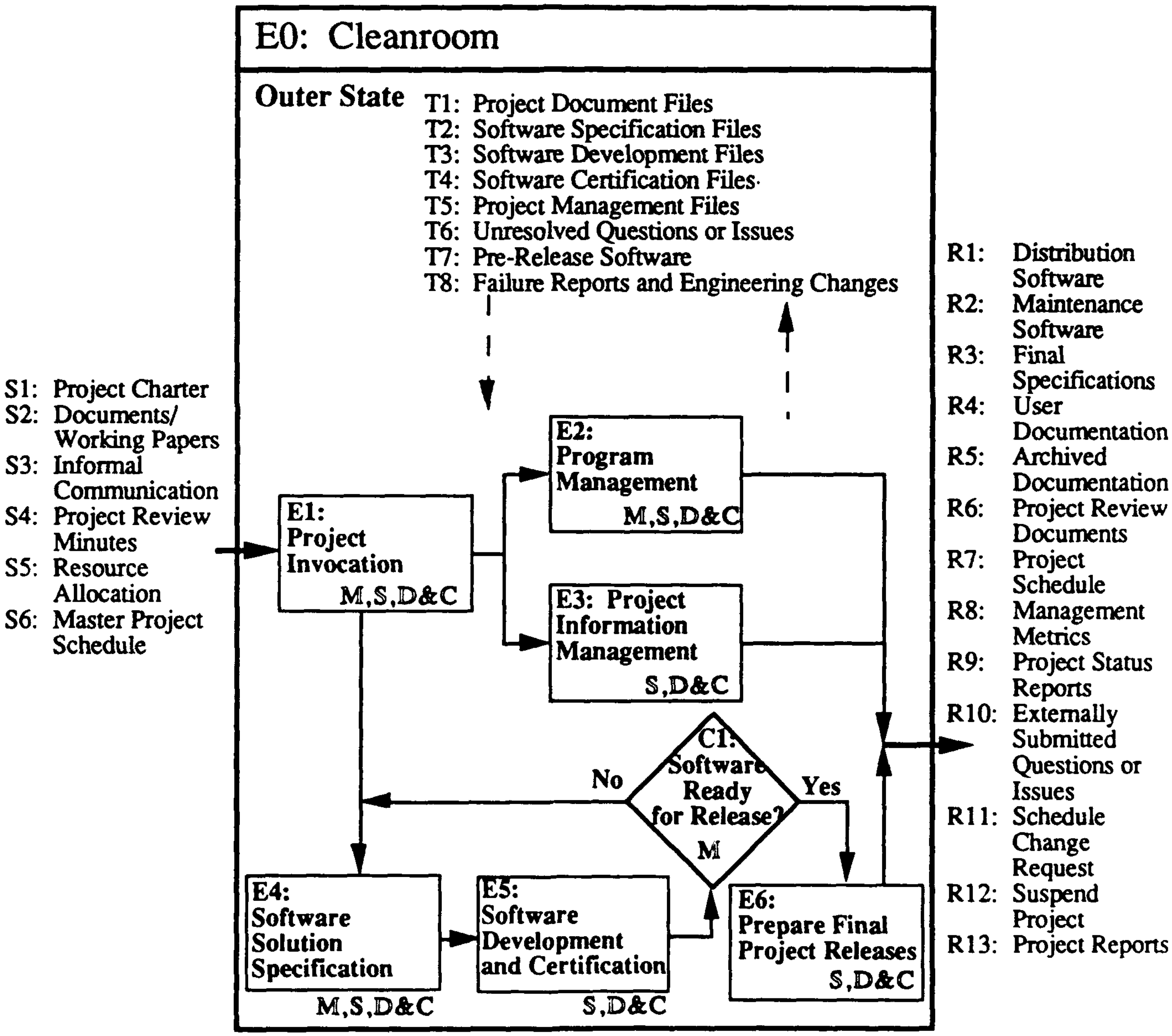
Keyword References Selected words that deserve additional explanation are listed in this section along with references to a more detailed description of the term or to the description of the process defined by the term.

In this section, each sub-section is started on a new page so it is easy for teams or managers to reproduce a section or sections to support work on a specific process.

Section 4.0 - Process E0 - Cleanroom

Process Summary The Cleanroom process clear box is the most general description of Cleanroom. Specifically, the project is invoked, then the program and externally received information are managed, a solution is specified and software is constructed and certified. Finally, the final project releases are put together. The process is illustrated in Figure 4.0.1, and is also described in greater detail below.

Figure 4.0.1 Cleanroom Process


E0: Cleanroom
Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
S1: Project Charter
S2: Documents/ Working Papers
S3: Informal Communication
S4: Project Review Minutes
S5: Resource Allocation
S6: Master Project Schedule
E1: Project Invocation M,S,D&C
E2: Program Management M,S,D&C
E3: Project Information Management S,D&C
C1: Software Ready for Release? M
No
Yes
E4: Software Solution Specification M,S,D&C
E5: Software Development and Certification S,D&C
E6: Prepare Final Project Releases S,D&C
R1: Distribution Software
R2: Maintenance Software
R3: Final Specifications
R4: User Documentation
R5: Archived Documentation
R6: Project Review Documents
R7: Project Schedule
R8: Management Metrics
R9: Project Status Reports
R10: Externally Submitted Questions or Issues
R11: Schedule Change Request
R12: Suspend Project
R13: Project Reports

Process Completed By Engineering team manager and Specification, Development and Certification teams.

Previous Process None - Process not started

Pre-condition None

Subsequent Processes None - Process completed

Stimuli Project Charter (S1), Documents/Working Papers (S2), Informal Communications (S3), Project Review Minutes (S4), Resource Allocation (S5), Master Project Schedule (S6).

Responses Distribution Software (R1), Maintenance Software (R2), Final Specifications (R3), User Documentation (R4), Archived Documentation (R5), Project Review Documents (R6), Project Schedule (R7), Management Metrics (R8), Project Status Reports (R9), Externally Submitted Questions or Issues (R10), Schedule Change Request (R11), Suspend Project (R12), Project Reports (R13).

State Data Usage This process does not directly affect state data, although its sub-processes do. The effect of the sub-processes on state data will be detailed when the sub-processes are described.

Process Description Process E0: Cleanroom is performed by following the algorithm below until the completion conditions are achieved:

```
use E1: Project Invocation
    con
        use E2: Program Management
        use E3: Project Information Management
        do
            do
                use E4: Software Solution Specification
                use E5: Software Development and Certification
            until
                C1: Software Ready for Release?
            od
            use E6: Prepare Final Project Releases
        od
    noc
```

C1: Software Ready for Release is a decision made by the engineering team manager. When the manager decides, as a result of available information (reliability results, code, etc.) that the project is ready, the preparation of the final product releases can begin. If the manager decides that the project is not ready, then the Construction Plan will be redone and Development and Certification will recommence.

E2: Program Management
E3: Project Information Management
E4: Software Solution Specification
E5: Software Development and Certification
are discussed in Sections 4.2, 4.3, 4.4 and 4.5, respectively.

Measurement Data Generated Effort

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

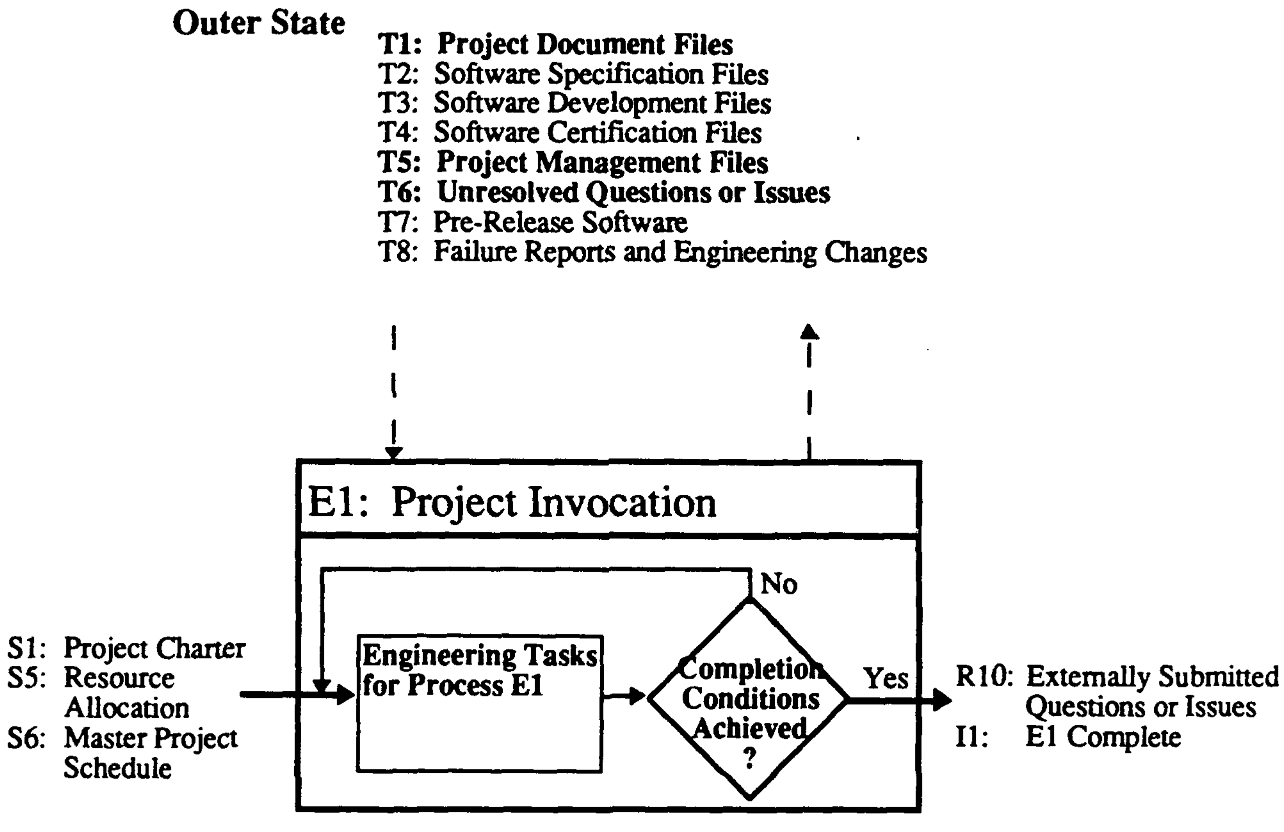
1. Have all necessary measurements been made?
2. Have all completion conditions for all six sub-processes (E1, E2, E3, E4, E5 and E6) been achieved?
3. Have all information to be preserved been placed in the correct state data?

Keyword References None

Section 4.1 - Process E1 - Project Invocation

Process Summary The Project Invocation process organizes and allocates the initial staff and resources for the project. The software development plan is created, and the initial project schedule is defined. Additionally, the initial Project Charter is determined. Finally, any questions or issues which arise that cannot be solved by the engineering team are externally submitted to be resolved. The process is illustrated in Figure 4.1.1, and is also described in greater detail below.

Figure 4.1.1 Project Invocation Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E1: Project Invocation
S1: Project Charter
S5: Resource Allocation
S6: Master Project Schedule
Engineering Tasks for Process E1
Completion Conditions Achieved ?
No
Yes
R10: Externally Submitted Questions or Issues
I1: E1 Complete

Process Completed By Engineering team manager and Specification, Development and Certification teams.

Previous Process Cleanroom (E0)

Pre-condition None

Subsequent Processes Program Management (E2), Project Information Management (E3), Software Solution Specification (E4).

Stimuli Project Charter (S1), Resource Allocation (S5), Master Project Schedule (S6).

Responses E1 Complete (I1), Externally Submitted Questions or Issues (R10).

State Data Usage This process results in material appearing in the Project Management Files (the Project Charter and the initial project schedule), and Unresolved Questions and Issues (initial questions or issues pertaining to the project). Additionally, the software development plan is the first item to appear in the Project Document Files as a working paper.

Process Description Complete (achieve the completion conditions for) the following engineering tasks during the Project Invocation process. They can be performed concurrently.

1. Allocate and organize the initial staff for the project.
2. Allocate and organize other initial resources for the project.
3. Agree on the initial Project Charter.
4. Create the software development plan, a working paper which outlines the software engineering procedures to be used for the project. This document or a prior software development plan can serve as a basis for this plan. The plan needs to be accepted by management. This plan must be accepted by the engineering teams.
5. Develop the initial project schedule. This includes integration of the Master Project Schedule and initial staff, resource and task allocations for the project.
6. Ask any questions that come up, first attempting to resolve them internally and, if unsuccessful, submitting them externally.

Measurement Data Generated Effort, State Data Produced

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

1. Have all initial staff members been put on a team?
2. Has the initial Project Charter been accepted?
3. Has the software development plan been created and accepted?
4. Has the initial project schedule been created?
5. Have initial task assignments been given?
6. Have all additions to, deletions from and modifications of the state data been completed?
7. Have all necessary measures been gathered?
8. Have all necessary responses been submitted?
9. Have all pertinent reviews for this process been completed?
10. Have all action items generated during reviews that pertain to this process been completed?
11. Have all information to be preserved been placed in the correct state data?

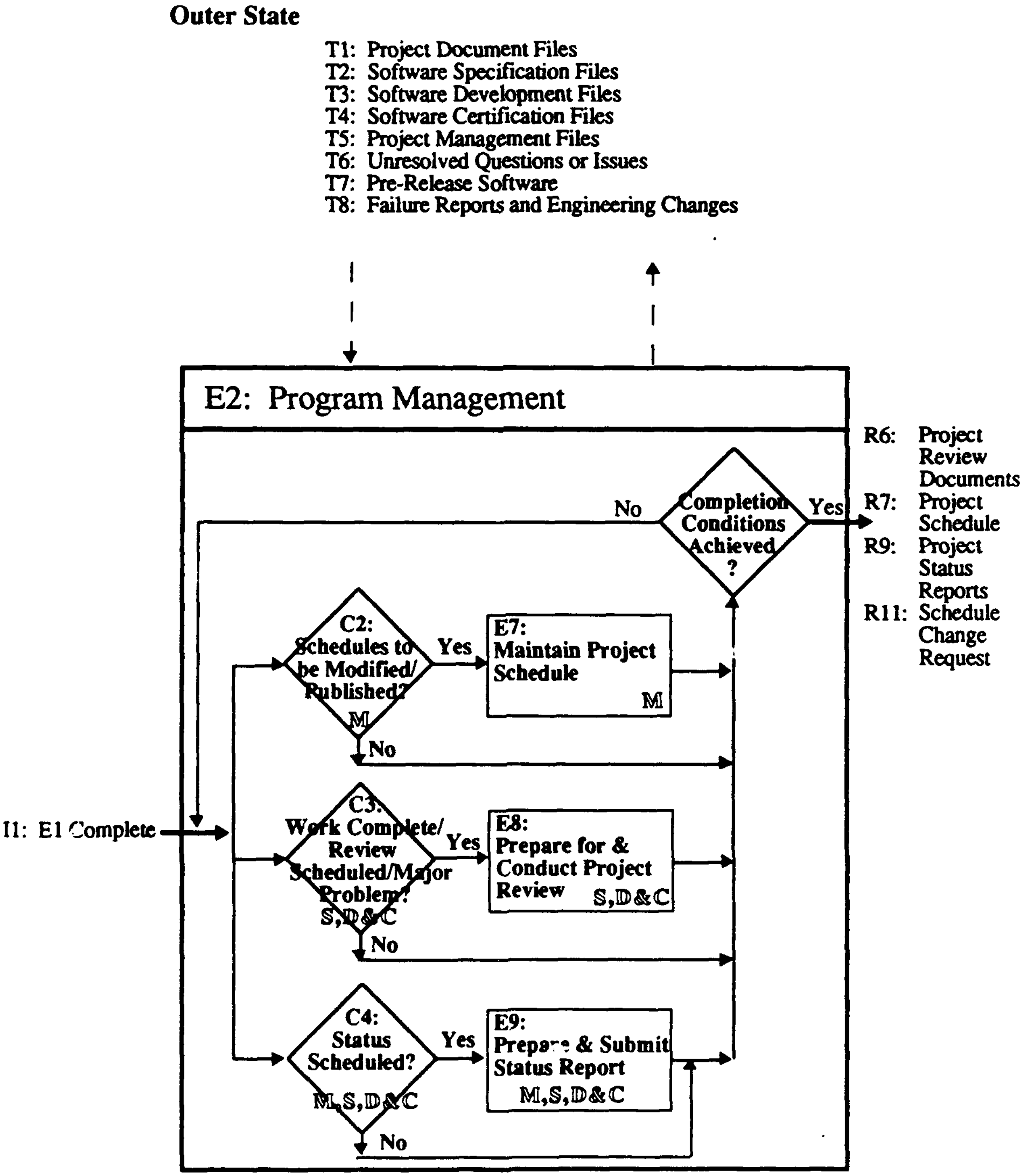
Keyword References

Review - Section 4.8
Unresolved Questions or Issues - Section 5.6

Section 4.2 - Process E2 - Program Management

Process Summary The Program Management process centralizes interaction with persons outside the engineering staff. During this process, the project schedule is maintained, requests to change the Master Project schedule are made, reviews are planned and conducted, and status reports are submitted. The process is illustrated in Figure 4.2.1, and is also described in greater detail below.

Figure 4.2.1 Program Management Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E2: Program Management
Completion Conditions Achieved ?
No
Yes
R6: Project Review Documents
R7: Project Schedule
R9: Project Status Reports
R11: Schedule Change Request
C2: Schedules to be Modified/ Published? M
Yes
E7: Maintain Project Schedule M
No
I1: E1 Complete
C3: Work Complete/ Review Scheduled/Major Problem? S,D&C
Yes
E8: Prepare for & Conduct Project Review S,D&C
No
C4: Status Scheduled? M,S,D&C
Yes
E9: Prepare & Submit Status Report M,S,D&C
No

Process Completed By Engineering team manager and Specification, Development and Certification teams.

Previous Process Project Invocation (E1)

Pre-condition None

Subsequent Process None - Project completed

Stimuli E1 Completed (I1)

Responses Project Review Documents (R6), Project Schedule (R7), Project Status Reports (R9), Schedule Change Request (R11)

State Data Usage This process does not directly affect state data, although its sub-processes do. The effect of the sub-processes on state data will be detailed when the sub-processes are described.

Process Description Process E2: Program Management is performed by following the algorithm below:

```
do
    con
        if C2: Schedules to be Modified/Published?
            then
                use E7: Maintain Project Schedule
        fi
        if C3: Work Complete/Review Scheduled/Major Problem?
            then
                use E8: Prepare for & Conduct Project Review
        fi
        if C4: Status Scheduled?
            then
                use E9: Prepare & Submit Status Report
        fi
    noc
until
    Completion Conditions achieved
od
```

C2: Schedules to be Modified/Published is a decision that is made by the project manager. The engineering team manager may receive a new Master Project Schedule, determine that the project schedule needs to be changed, or the calendar may reach a time when the project schedule needs to be published (due to information in the Project Charter or Master Project Schedule). These are changes to the external schedule. The engineering team manager may also decide that a task assignment needs to be added, modified or deleted. These are changes to the internal schedule. If any of these cases occur, then process E7: Maintain Project Schedule is invoked.

C3: Work Complete/Review Scheduled/Major Problem is a decision made by the engineering team and management. The engineering team may decide that a certain amount of work is completed to warrant an external review. The team may also determine that there is a major problem that they cannot solve and that needs to be addressed by management. Finally, a review schedule may be part of the Master Project Schedule or a review may be called by management, for which the project team must prepare. If any of these cases are true, then process E8: Prepare for & Conduct Project Review is invoked.

C4: Status Scheduled is based on a schedule determined by the organization, or appears as a part of the Project Charter. If a status report is scheduled, then it must be submitted, which means process E9: Prepare and Submit Status Report is invoked.

E7: Maintain Project Schedule
E8: Prepare For and Conduct Review
E9: Prepare and Submit Project Status Reports
E9: Prepare and Submit Status Reports
are discussed in Sections 4.2.7, 4.2.8 and 4.2.9, respectively

Measurement Data Generated Effort

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

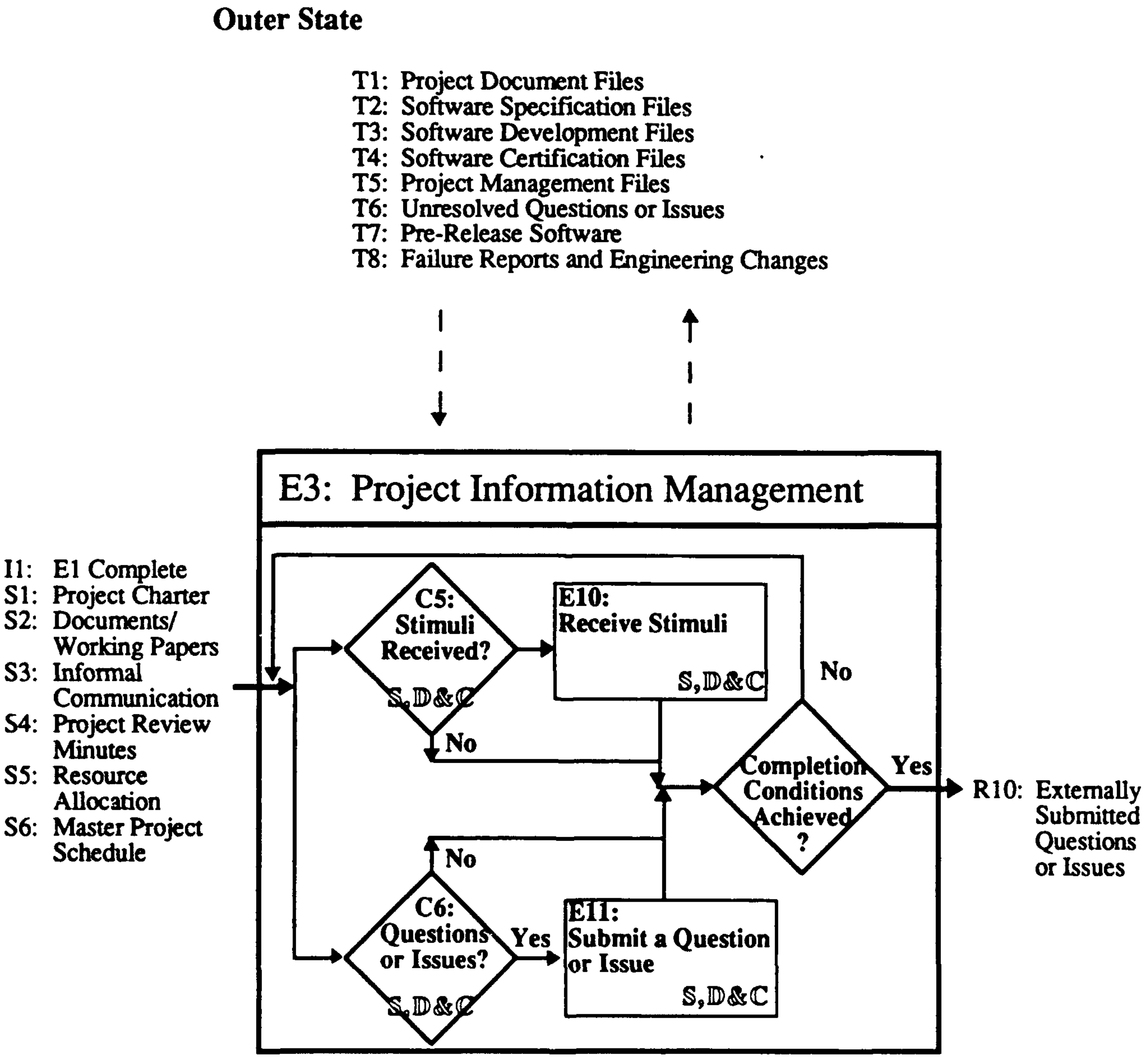
1. Have all necessary measures been gathered?
2. Have all completion conditions for all three subprocesses (E7, E8 and E9) been achieved?
3. Is the project completed?

Keyword References None

Section 4.3 - Process E3 - Project Information Management

Process Summary The Project Information Management process is concerned with the receipt and filing of all externally received stimuli. The stimuli are filed in a manner which makes the information easily accessible by the entire staff. Additionally, questions or issues which are to be submitted internally or externally or are to be resolved internally are organized in this process, since this is where the responses to those questions will appear. The process is illustrated in Figure 4.3.1, and is also described in greater detail below.

Figure 4.3.1 Project Information Management Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E3: Project Information Management
I1: E1 Complete
S1: Project Charter
S2: Documents/ Working Papers
S3: Informal Communication
S4: Project Review Minutes
S5: Resource Allocation
S6: Master Project Schedule
C5: Stimuli Received?
S,D&C
E10: Receive Stimuli
S,D&C
No
No
Completion Conditions Achieved ?
Yes
R10: Externally Submitted Questions or Issues
No
C6: Questions or Issues?
S,D&C
Yes
E11: Submit a Question or Issue
S,D&C

Process Completed By Specification, Development and Certification teams.

Previous Process Project Invocation (E1)

Pre-condition None

Subsequent Process None - Project completed

Stimuli E1 Complete (I1), Project Charter (S1), Documents/Working Papers (S2), Informal Communication (S3), Project Review Minutes (S4), Resource Allocation (S5), Master Project Schedule (S6).

Responses Externally Submitted Questions or Issues (R10)

State Data Changes This process does not directly affect state data, although its sub-processes do. The effect of the sub-processes on state data will be detailed when the sub-processes are described.

Process Description Process E3: Project Information Management is performed by following the algorithm:

```
do
    con
        if C5: Stimuli Received?
            then
                use E10: Receive Stimuli
        fi
        if C6: Questions or Issues?
            then
                use E11: Submit a Question or Issue
        fi
    noc
until
    Completion Conditions achieved
od
```

C5: Stimuli Received is true when any external stimuli is given to the engineering team. When it is true, process E10: Receive Stimuli is invoked.

C6: Questions or Issues is true when any project staff member (Specification, Development or Certification) determines, at any time during the project, that there is a question or issue that the project staff member cannot resolve. When C6 is true, process E11: Submit a Question or Issue is invoked.

E10: Receive Stimuli
E11: Submit a Question or Issue
are described in Sections 4.10 and 4.11, respectively.

Measurement Date Generated Effort

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

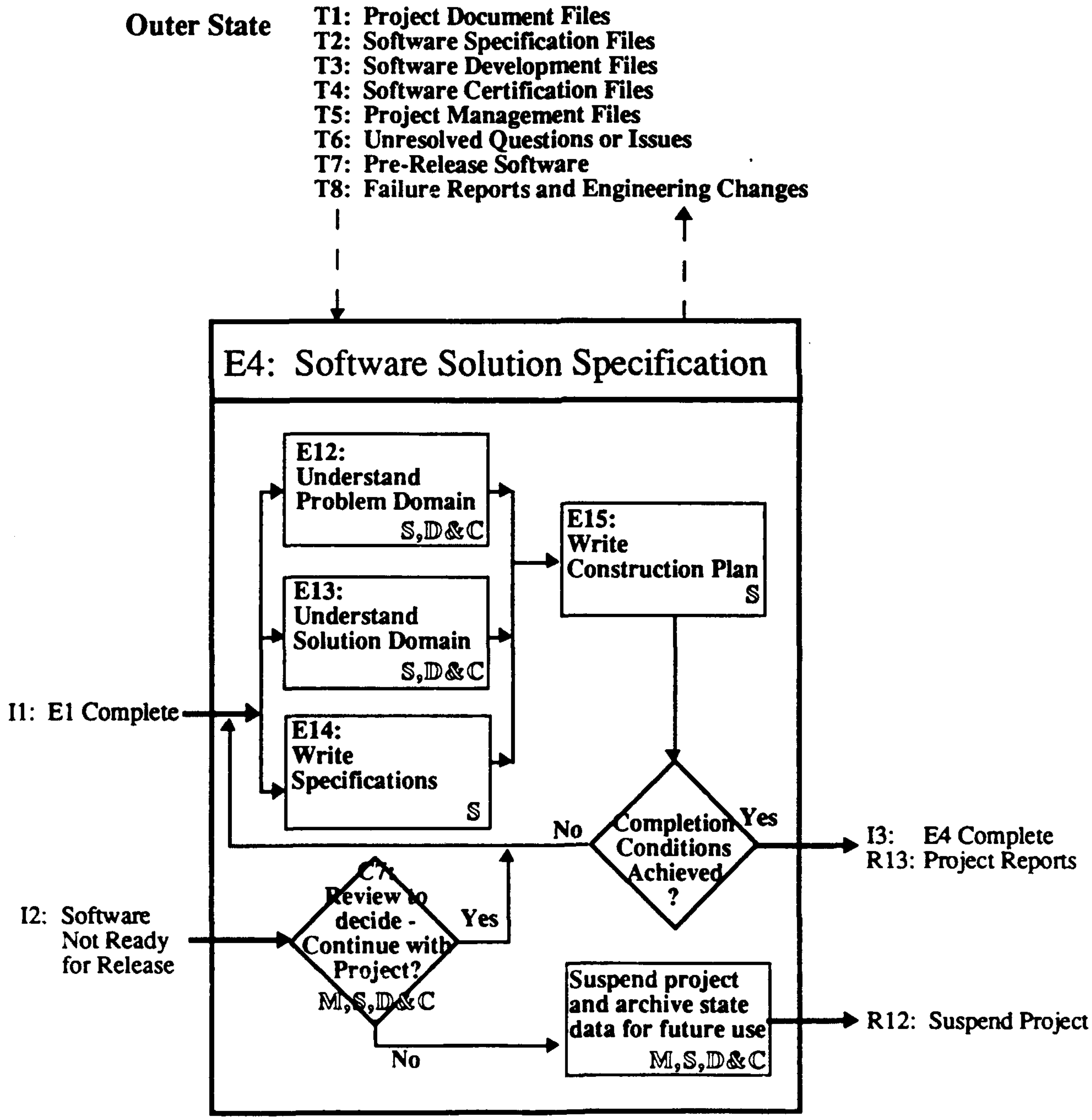
1. Have all necessary measures been gathered?
2. Have all completion conditions for both subprocesses (E10 and E11) been achieved?
3. Is the project completed?

Keyword References None

Section 4.4 - Process E4 - Software Solution Specification

Process Summary The Software Solution Specification process focuses on understanding the problem to be solved and specifying a solution. In addition to understanding the problem domain, the engineering staff must also understand the solution domain, while the Specification team must write the Specifications and the Construction Plan. Additionally, if a project must be re-planned or re-done, the decision to suspend the project is also made in this process. The process is illustrated in Figure 4.4.1, and is also described in greater detail below.

Figure 4.4.1 Software Solution Specification Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E4: Software Solution Specification
E12: Understand Problem Domain S,D&C
E13: Understand Solution Domain S,D&C
E14: Write Specifications S
E15: Write Construction Plan S
I1: E1 Complete
Completion Conditions Achieved ?
No
Yes
I3: E4 Complete
R13: Project Reports
I2: Software Not Ready for Release
C7: Review to decide - Continue with Project? M,S,D&C
Yes
No
Suspend project and archive state data for future use M,S,D&C
R12: Suspend Project

Process Completed By Engineering team manager and Specification, Development and Certification teams.

Previous Process Project Invocation (E1) or Software Development and Certification (E5)

Pre-condition None for first iteration, subsequent iterations have condition Software Ready for Release? (C1) = FALSE

Subsequent Process Software Development and Certification (E5)

Stimuli E1 Completed (I1), C1 False (I2)

Responses E4 Completed (I3), Suspend Project (R12), Project Reports (R13).

State Data Usage Since the review to decide whether to continue the project and the process of suspending the process and archiving state data are both a part of this process, all of the state data will be used as reference material for those tasks.

Process Description The following algorithm performs the E4: Solution Specification process:

```
do
    if (C1 not a stimulus)
        then
            con
                use E12: Understand Problem Domain
                use E13: Understand Solution Domain
                use E14: Write Specifications
            noc
            use E15: Write Construction Plan
        else
            if C7: Review to decide - Continue with Project
                then
                    con
                        use E12: Understand Problem Domain
                        use E13: Understand Solution Domain
                        use E14: Write Specifications
                    noc
                    use E15: Write Construction Plan
                else
                    Suspend Project (R12)
                    Entire engineering team staff will archive state data for future use
                    Exit process
            fi
    fi
until
    Completion Conditions achieved
od
```

C7: Review to decide - Continue with Project is an assessment of the status of the project. This is a major review that only occurs when the engineering team manager or upper management determine that the code fulfills the specification, but the specification is incorrect. At this review, the decision is made whether the project should be suspended, and the material should be archived for future use, or if the project should continue (by writing a new specifications document and continuing from there).

E12: Understand Problem Domain
E13: Understand Solution Domain
E14: Write Specifications
E15: Write Construction Plan
are described in Sections 4.12, 4.13, 4.14 and 4.15, respectively.

Measurement Data Generated Effort

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

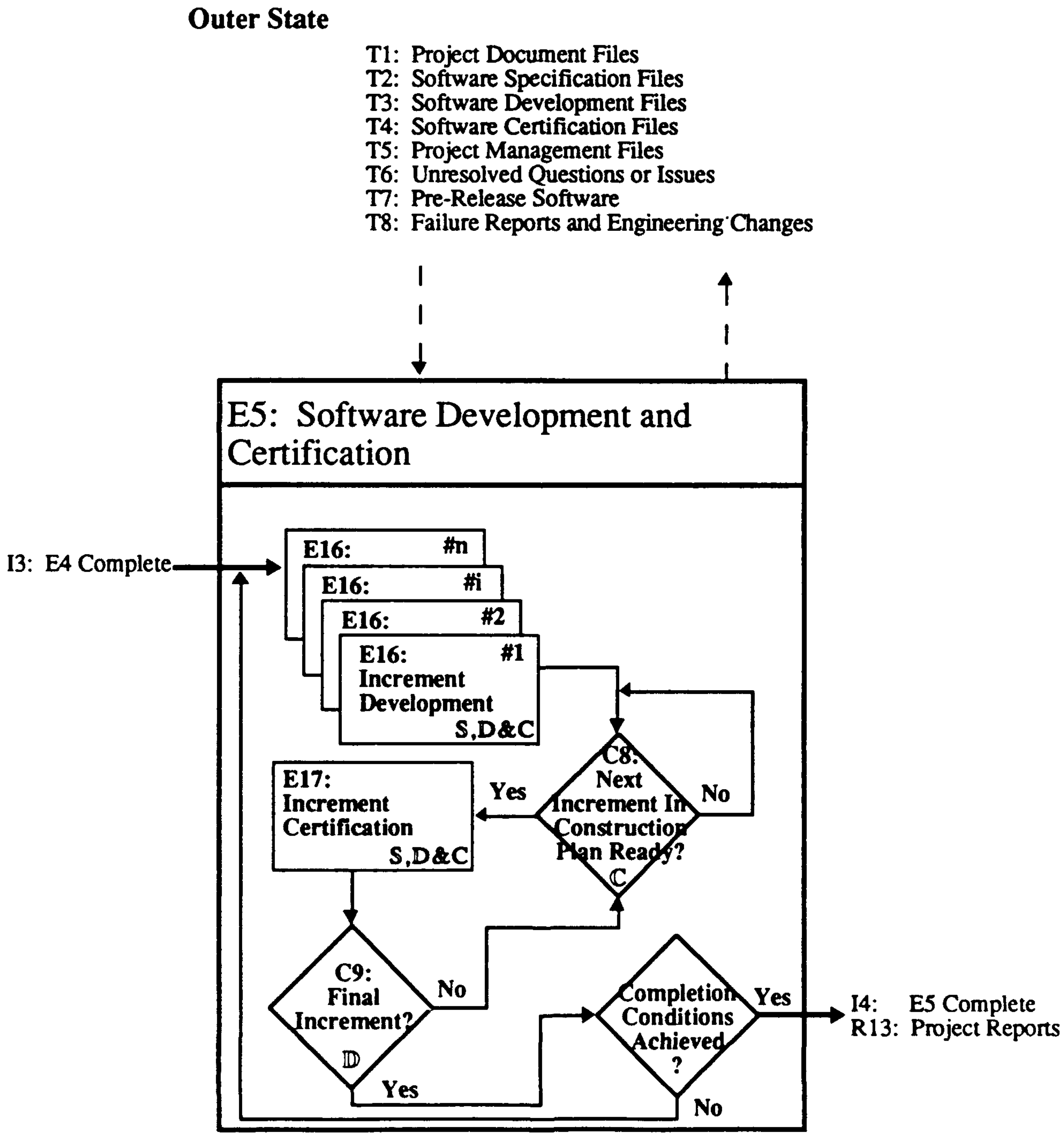
1. Have all necessary measures been gathered?
2. Have the completion conditions of all four sub-processes (E12, E13, E14, and E15) been achieved?
3. If the project was suspended, has all material been archived in a manner that will be useful to individuals who will need the material in the future?

Keyword References None

Section 4.5 - Process E5 - Software Development and Certification

Process Summary The Software Development and Certification process implements the solution which is documented in the Specifications. The sub-processes include incremental development and certification. The process is illustrated in Figure 4.5.1, and is also described in greater detail below.

Figure 4.5.1 Software Development and Certification Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E5: Software Development and Certification
I3: E4 Complete
E16: #n
E16: #i
E16: #2
E16: #1 Increment Development S,D&C
E17: Increment Certification S,D&C
Yes
C8: Next Increment In Construction Plan Ready? C
No
C9: Final Increment? D
No
Yes
Completion Conditions Achieved ?
Yes
No
I4: E5 Complete
R13: Project Reports

Process Completed By Specification, Development and Certification teams.

Previous Process Software Solution Specification (E4)

Pre-condition None

Subsequent Process Prepare Final Project Releases (E6) or Software Solution Specification (E4)

Stimuli E4 Complete (I3)

Responses E5 Complete (I4), Project Reports (R13)

State Data Usage This process does not directly affect state data, but its sub-processes do. The effect of the sub-processes on state data will be discussed when the sub-processes are described.

Process Description Use the algorithm below to perform process E5: Software Development and Certification.

```
do
    con
        use E16: Increment Development (for increment 1)
        use E16: Increment Development (for increment 2)
        ...
        use E16: Increment Development (for increment i)
        ...
        use E16: Increment Development (for increment n)
        while C8: Next Increment in Construction Plan Ready?
            do
                use E17: Increment Certification
                if C9: Final Increment?
                    then
                        Exit process
                fi
            od
     noc
until
    Completion Conditions achieved
od
```

C8: Next Increment in Construction Plan Ready is true when the code for the next increment, as dictated in the Construction Plan (Volume 6 of the Specifications), is ready to be submitted to the Certification team from the Development team. Certification for an increment cannot occur until the next increment for the system to be certified is verified and therefore ready for Certification.

C9: Final Increment is true if the final increment has been integrated into the system and has been certified.

E16: Increment Development
E17: Increment Certification
are described in Sections 4.16 and 4.17, respectively.

Measurement Data Generated Effort

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

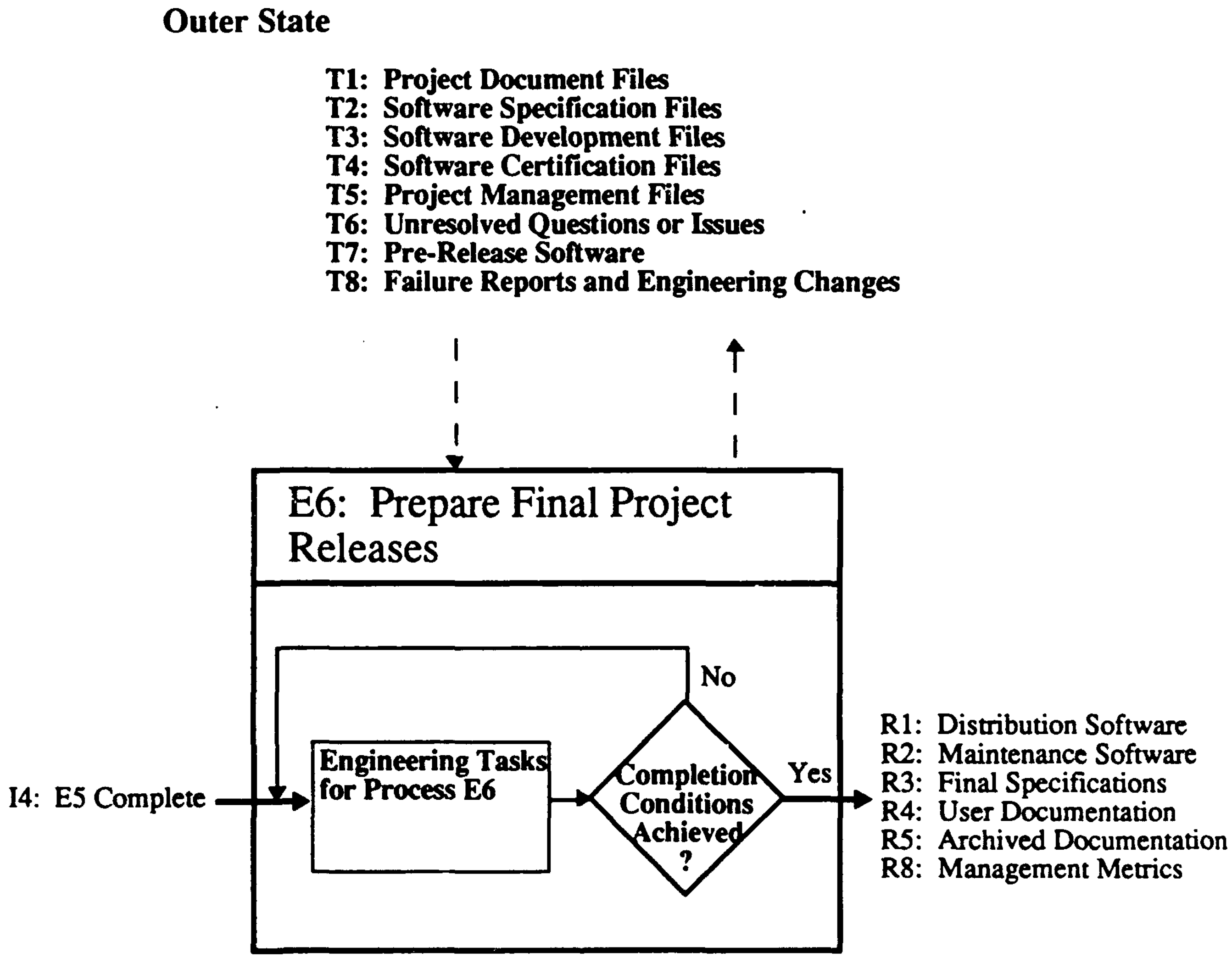
1. Have all necessary measures been gathered?
2. Have all completion conditions from both subprocesses (E16 and E17) been achieved (This is equivalent to asking whether all increments have been developed, verified and certified)?

Keyword References None

Section 4.6 - Process E6 - Prepare Final Project Releases

Process Summary The Prepare Final Project Releases process converts the state data into the set of final products for the customer. The process is illustrated in Figure 4.6.1, and is also described in greater detail below.

Figure 4.6.1 Prepare Final Project Releases Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E6: Prepare Final Project Releases
I4: E5 Complete
Engineering Tasks for Process E6
Completion Conditions Achieved ?
No
Yes
R1: Distribution Software
R2: Maintenance Software
R3: Final Specifications
R4: User Documentation
R5: Archived Documentation
R8: Management Metrics

Process Completed By Specification, Development and Certification teams.

Previous Processes Software Development and Certification (E5).

Pre-condition Software Ready for Release? (C1) = TRUE

Subsequent Process None

Stimuli E5 Complete (I4)

Responses Distribution Software (R1), Maintenance Software (R2), Final Specifications (R3), User Documentation (R4), Archived Documentation (R5), Management Metrics (R8).

State Data Usage This process does not modify the state data, but it uses all of the state data available as reference material in order to generate responses, which are the final project releases. The Project Management Files, specifically the project schedule, is also used to determine an allocation of effort and resources for this process.

Process Description Prepare deliverables using all available resources. These activities may be completed concurrently and continue until the completion conditions are achieved.

1. Distribution Software (R1) and Maintenance Software (R2) are based on the Pre-Release Software (T7).
2. The Final Specifications (R3) is based on the Software Specification Files (T2).
3. User Documentation (R4) is based on Software Development Files (T3) and Software Specification Files (T2).
4. Archived Documentation (R5) is based on Project Document Files (T1), Software Development Files (T3), Software Certification Files (T4), Project Management Files (T5), Unresolved Questions or Issues (T6), and Failure Report and Engineering Changes (T8).
5. Management Metrics (R8) is based on Software Specification Files (T2), Software Development Files (T3), Software Certification Files (T4), Project Management Files (T5), and Unresolved Questions or Issues (T6).
6. The engineering team may also need to participate in a review where the sponsor accepts the deliverables (for example, if the project is following 2167A, the reviews are called the Formal Qualification Review and the Production Readiness Review).

Measurement Data Generated Effort

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

1. Are all deliverables complete and consistent?
2. Have all necessary measures been gathered?
3. Have all additions to, deletions from and modifications to the state data been completed?
4. Have all responses from the process been submitted?
5. Have all pertinent reviews for this process been completed?
6. Have all action items generated during reviews that pertain to this process been completed?

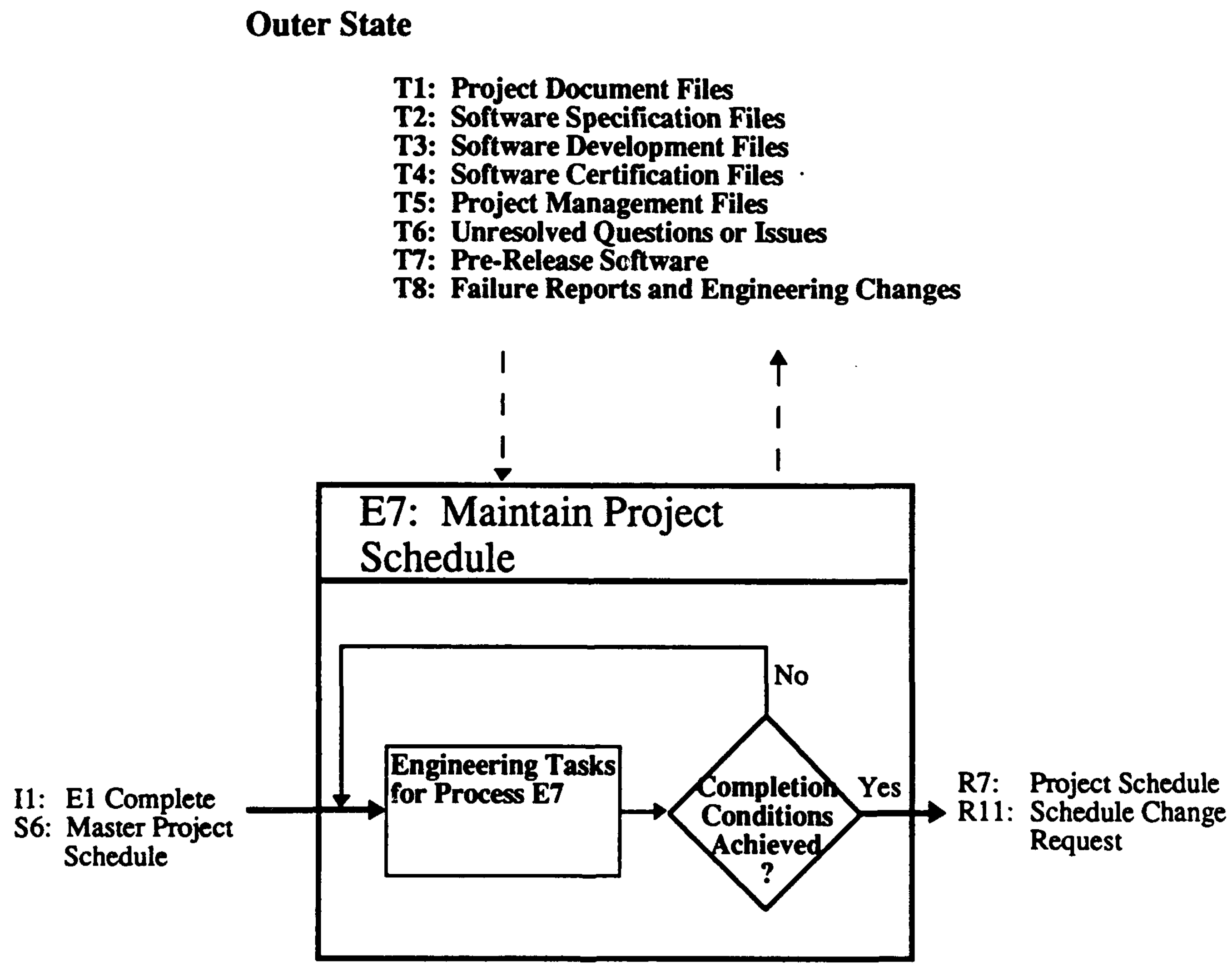
Keyword References

Review - Section 4.8

Section 4.7 - Process E7 - Maintain Project Schedule

Process Summary The Maintain Project Schedule process updates the project schedule, which is a part of the Project Management Files, to have it accurately reflect the past, present and future of the project. The schedule lists tasks, and for each task there are dates and resource and effort allocations. Additionally, all external and internal milestones and reviews are maintained here. The process is illustrated in Figure 4.7.1, and is also described in greater detail below.

Figure 4.7.1 Maintain Project Schedule Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E7: Maintain Project Schedule
No
I1: E1 Complete
S6: Master Project Schedule
Engineering Tasks for Process E7
Completion Conditions Achieved ?
Yes
R7: Project Schedule
R11: Schedule Change Request

Process Completed By Engineering team manager.

Previous Processes Program Management (E2)

Pre-condition Schedule to be Modified/Published? (C2) = TRUE

Subsequent Process Maintain Project Schedule (E7)

Stimuli E1 Complete (I1), Master Project Schedule (S6)

Responses Project Schedule (R7), Schedule Change Request (R11).

State Data Usage This process modifies E5, the Project Management Files. Specifically, the project schedule and task assignments in the Project Management Files are modified. That state data will be changed, while all other state data is just used as reference.

Process Description Complete (achieve Completion Conditions for) the following engineering tasks each time the process is invoked as a result of condition C2 being true.

1. Review state data in order to determine the present state of the project.
2. Based on the present state of the project, determine engineering tasks that must be added, modified or deleted.
3. For each task, determine the time period necessary for the task, and assign staff and resources to the task. Complete a task assignment for the task. If a task assignment is to be deleted, do so. Also modify any existing tasks that must be modified. Update the project schedule according to the task assignments.
4. Include task assignments to fulfill any reviews, milestones, or other items on the Master Project Schedule. Note all reviews, milestones or other items on the project schedule.
5. If the any new or modified task assignment make the present project schedule inconsistent with the Master Project Schedule, submit a Schedule Change Request (R11). In these cases, the project schedule must not be changed until a new/updated Master Project Schedule is received. If a new Master Project Schedule is not received, then the change request has been denied and the engineering team must conform to the previous schedule.
6. The project schedule is published whenever there is a change to the Master Project Schedule, and at the intervals stated in the Project Charter and the Master Project Schedule.

Measurement Data Generated Effort, Status or Schedule

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process each time the process is invoked as a result of condition C2 being true.

1. Have all defined task assignments been given a completion schedule, as well as staff and resource allocations?
2. Are all task assignments (internal schedule) consistent with the project schedule and the Master Project Schedule (external schedule)?
3. Have all known milestones been put on the schedule?
4. Is the project schedule complete and consistent in relationship to the Master Project Schedule?
5. Has the project schedule been published due to changes in the Master Project Schedule or by time interval?
6. Does the project schedule list all sponsor requested reviews?

7. Have all additions to, deletions from and modifications of the state data been made?
8. Have all necessary measures been gathered?
9. Have all external responses been submitted?
10. Have all pertinent reviews for this process been completed?
11. Have all action items generated during reviews that pertain to this process been completed?
12. Have all information to be preserved been placed in the correct state data?

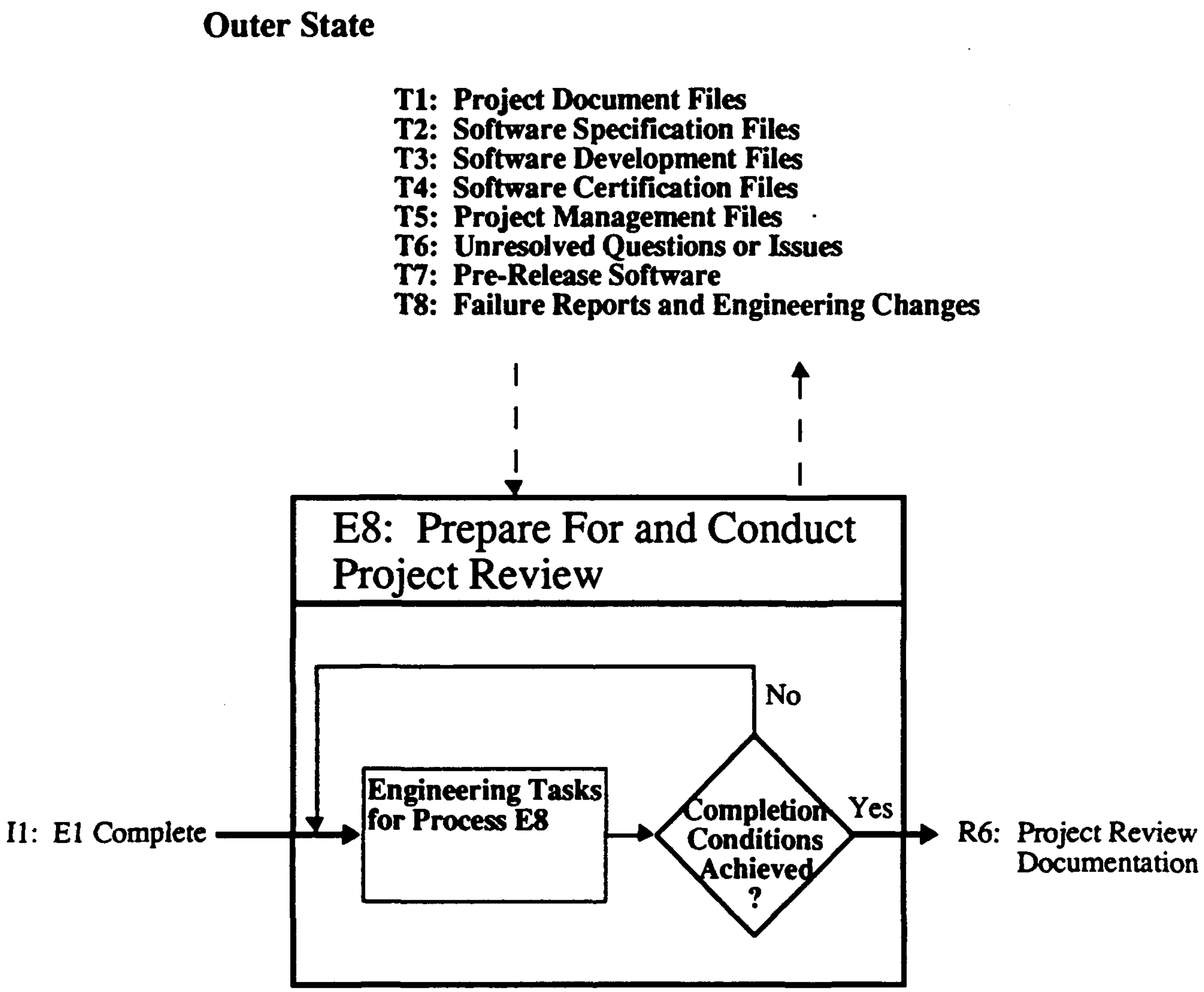
Keyword References

Review - Section 4.8

Section 4.8 - Process E8 - Prepare For and Conduct Project Review

Process Summary The Prepare For and Conduct Project Review process gives the engineering team the opportunity to gather materials, distribute them and conduct a project review. Reviews are conducted when one is scheduled, a certain amount of work is completed, or when a major problem is evident. The process is illustrated in Figure 4.8.1, and is also described in greater detail below.

Figure 4.8.1 Prepare For and Conduct Project Review Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E8: Prepare For and Conduct Project Review
No
I1: E1 Complete
Engineering Tasks for Process E8
Completion Conditions Achieved ?
Yes
R6: Project Review Documentation

Process Completed By Specification, Development and Certification teams.

Previous Processes Program Management (E2)

Pre-condition Work Complete/Review Scheduled/Major Problem? (C3) = TRUE

Subsequent Process Program Management (E2)

Stimuli E1 Complete (I1)

Responses Project Review Documentation (R6).

State Data Usage This process uses the state data, to generate the review material. All of the state data is used to prepare for and conduct a review.

Process Description Reviews are processes, requested by the project sponsor, in order to assess progress on a project. A project may have any number of reviews, including none. There are a number of reviews in which the engineering team typically will be required to participate. These reviews typically include:

a) Software Requirements Review (where the understanding of the mission of the software project and available materials are audited)
b) Preliminary Design Review (where the Specification are audited),
c) Critical Design Review (where each increment, both development and certification, is audited),
d) Software Acceptance Review (which checks for the completeness of products before delivery),
e) Increment Failure Review (where failure-laden increments are audited).

Although each review has a different purpose, each (as well as unscheduled reviews) is conducted in the same manner, using the following sequence of engineering tasks each time the process is invoked as a result of condition C3 being true. The processes are conducted until the Completion Conditions are achieved:

Prepare for a review by using the following approach:

1. Assembling material necessary for the review from the current state data files.
2. Checking the material to ensure that it is complete, consistent and accurate.
3. Confirming date for review. If review is as yet unscheduled, the project schedule must be changed.

Conduct reviews using the following approach:

1. Distribute copies of material to be reviewed in advance.
2. Give review team ample time to review all documents.
3. Give the review team ample opportunity to prepare responses.
4. Be present at the scheduled review.
5. Deliver a complete, clear and concise presentation.
6. Answer all questions asked by the review team, and record unresolved questions.
7. Participate in briefing of review team.
8. Record all action items in minutes with a timetable outlining response schedule for the action items. Specifically, each action item needs to be delegated to a person or a team. The action item must also be assigned a date by which it needs to be resolved.

Measurement Data Generated Effort, State Data Produced

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

1. Has the Engineering team manager determined that the Specification, Development and Certification teams are sufficiently prepared for the project review?
2. Have all additions to, deletions from and modifications of the state data been completed?
3. Have all external responses been submitted?
4. Have all necessary measures been gathered?
5. Is the sponsor satisfied that the review is complete? If not, has another review been scheduled?
6. Have all action items from the review been assigned?

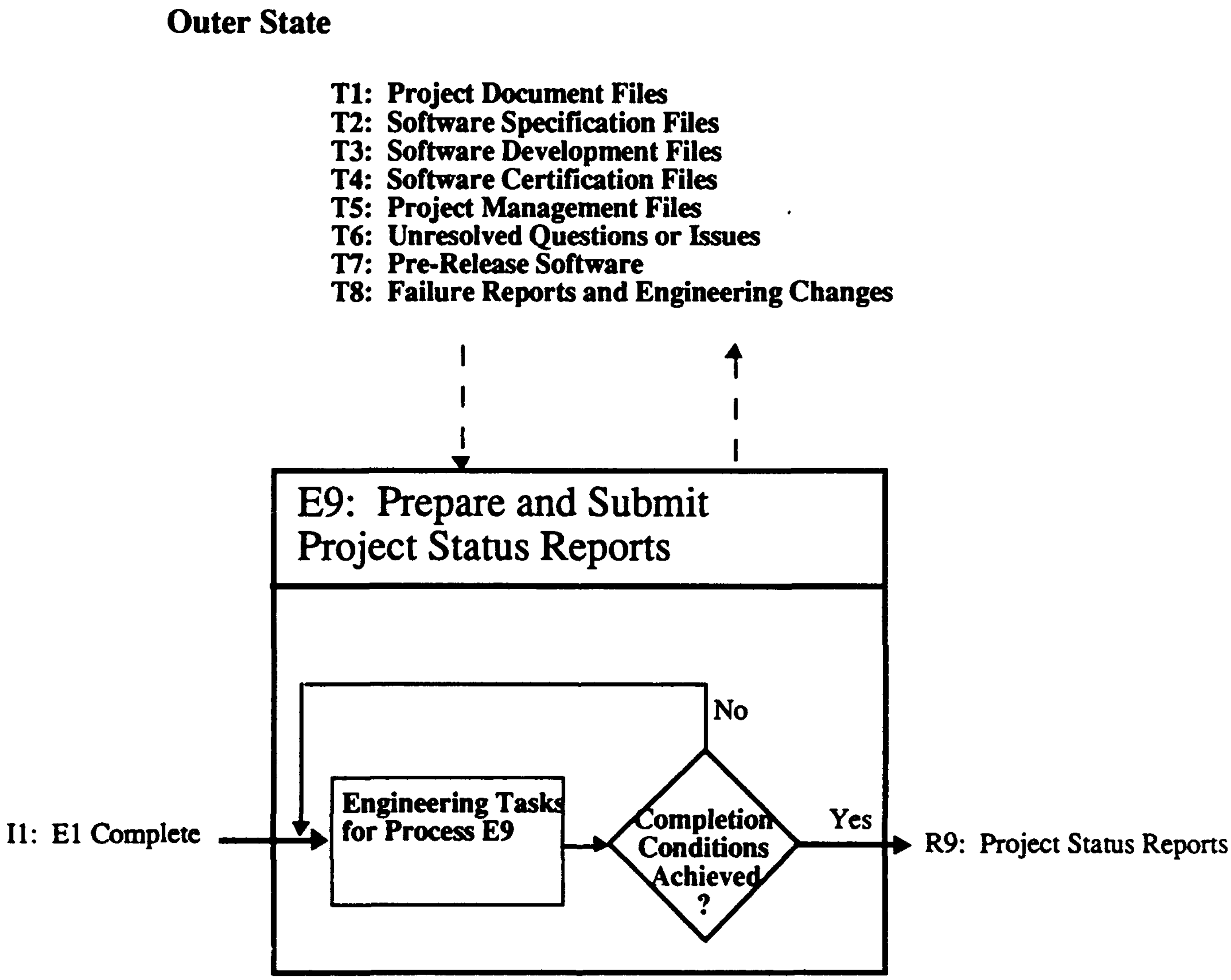
Keyword References

Review - Section 3.10

Section 4.9 - Process E9 - Prepare and Submit Project Status Reports

Process Summary The Prepare For and Submit Project Status Reports process leads to the preparation and submission of status reports, as the schedule dictates. The primary task is the collection and organization of measurement data, which is summarized in the status reports. The process is illustrated in Figure 4.9.1, and is also described in greater detail below.

Figure 4.9.1 Prepare For and Submit Project Status Reports Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E9: Prepare and Submit Project Status Reports
I1: E1 Complete
Engineering Tasks for Process E9
Completion Conditions Achieved ?
No
Yes
R9: Project Status Reports

Process Completed By Engineering team manager and Specification, Development and Certification teams.

Previous Processes Program Management (E2)

Pre-condition Status Scheduled? (C4) = TRUE

Subsequent Process Program Management (E2)

Stimuli E1 Complete (I1)

Responses Project Status Reports (R9)

State Data Usage This process modifies T2 - T6 by inserting new information, as well as using them as reference material. Other state data are not modified, they are used solely as reference material. The project schedule of the Project Management Files is used to determine when status reports should be submitted, and to determine an effort and resource allocation for the process.

Process Description Gather all necessary data, until Completion Conditions are achieved, to maintain intellectual control over the development process. Data gathered during other processes are submitted during this process. Available resources are used to complete and submit Project Status Reports, according to the schedule dictated in the project schedule. The specific types of status reports to be submitted are (should be) described in the Master Project Schedule.

The following is a basis set of metrics that need to be gathered for a Cleanroom project. These can be augmented by other (usually organization specific) measures. The actual forms or other schema for data collection, and a suggested strategy for gathering data are also presented here. Depending on the directions in the Project Charter, some, most or all of this information will be submitted on status reports. The data is categorized into six general areas. Units represent the specific information that is gathered. The rate suggests how often the data should be gathered. The gathered during column lists the processes during which the specific data is can be found. Data is stored in the state data listed below. The goal here is to gather as much useful data as possible while expending the minimum effort necessary. The data to be gathered is as follows:

Type of Data	Unit(s) Gathered	Rate	During	Stored In
Effort	Hours per process	Work Sampling	E0 - E24	T5
State Data Produced	Boxes developed	Bi-weekly	E18	T3
	Test scenarios developed		E19	T4
	Questions/Issues submitted		E1, E11	T6
	Questions/Issues resolved		E1, E11	T6
	Specification changes made		E14, E15, E20	T2
Status or Schedule	Tasks assigned	Weekly	E7	T5
	Tasks completed		E7	T5
	Schedule modifications		E7	T5

Library Management	Numbers of components	By system version for each increment	E22	T4
	Sizes of components		E22	T4
	Numbers of new components		E22	T4
	Numbers of modified components		E22	T4
	Number of compiled components		E22	T4
	Compilation CPU time		E22	T4
	Number of assembled components		E22	T4
	System dates for each version		E22	T4
Reliability scenario	Execution time per test	By system version	E23	T4
	Failure report numbers per test scenario		E22, E23	T4
	System version for each test scenario		E22, E23	T4

In addition to the measures gathered here, a number of measures such as the following can be derived from the above-mentioned data.

Measures	Derived From	Rate	Stored In
Productivity	Hours per Process Sizes of components	By increment	T5
System Reliability	Dates and states of executable systems Execution time per test scenario Failure report numbers per test scenario System version for each test scenario	By system version	T4

Most of the library management and reliability measures can be gathered directly by simple on-line tools with a minimum of human effort. Effort, state data produced, and status/schedule are gathered directly by people, but should not lead to a large amount of extra expended effort. Engineering team members will want to have a complete count of state data produced, as it shows that they are completing some amount of work, and is data that is easily verifiable, as one will be able to cross reference boxes with executable modules. The manager always needs to keep track of task status and schedule, so the manager's additional effort is simply summarizing existing information, which most managers already do. Measuring effort by work sampling is a little more sophisticated. With work sampling, engineering team members are asked at random intervals, what they are doing. This leads to accurate measurement, as the team member only needs to remember what is being done at the present moment, and will lead to accurate extrapolation, using the power of statistics. Derived measures will need some additional effort, but that effort will be minimal.

These measures will lead to the accurate assessment of the Cleanroom process and product. They will product sufficient information to modify the process, if necessary, and still be useful in the modified process. They can also be very efficiently gathered.

Measurement Data Generated Effort

Completion Conditions

1. Are the Project Status Reports complete and accurate for the time period for which the status report is being submitted?
2. Are consistent criteria being used to complete the Project Status Reports?
3. Have all Project Status Reports for the time period been completed and submitted?
4. Have all additions to, deletions from and modifications of the state data been made?
5. Have all external responses been submitted?
6. Have all necessary measures been gathered?
7. Have all pertinent reviews for this process been completed?
8. Have all action items generated during reviews that pertain to this process been completed?

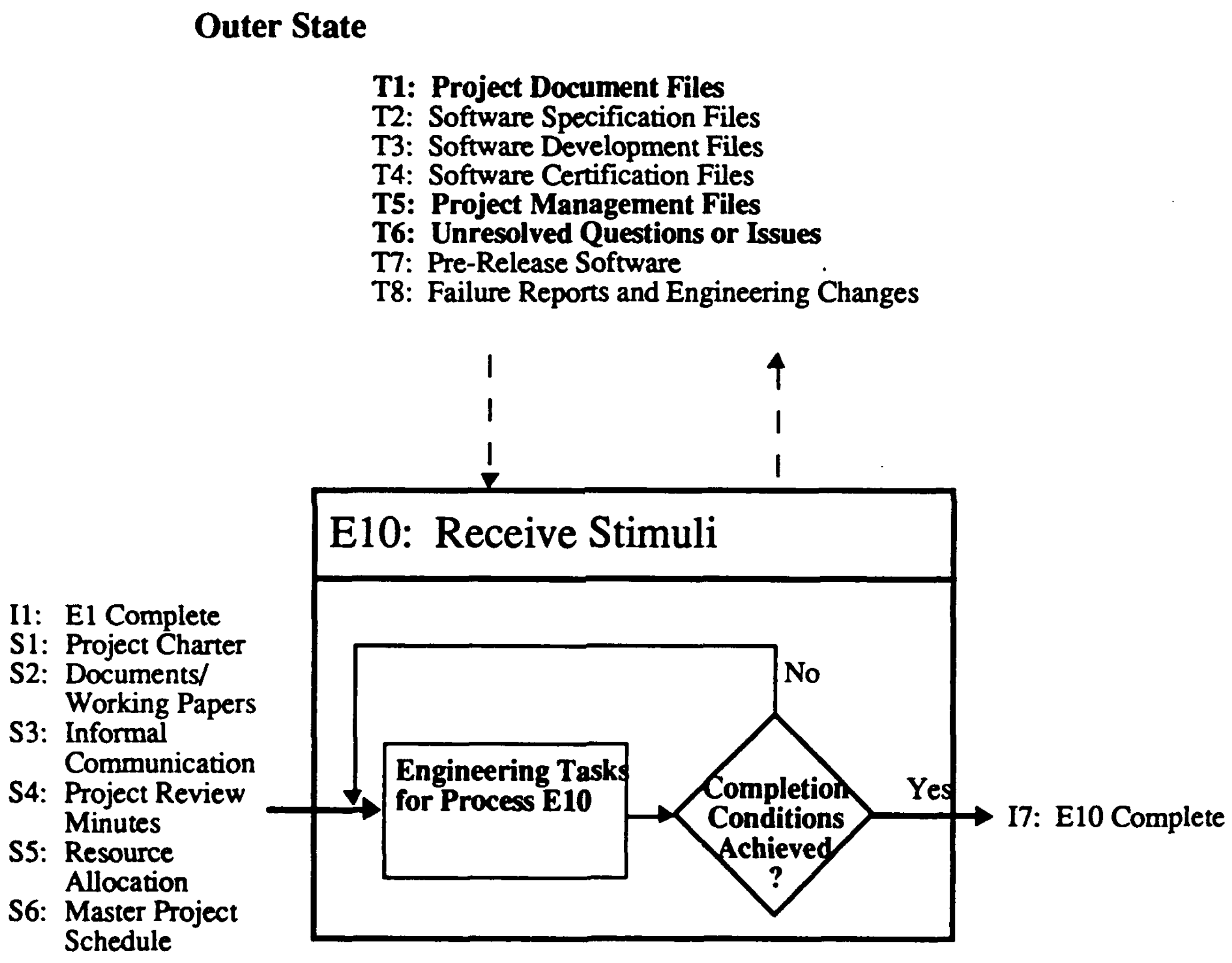
Keyword Reference

Review - Section 4.8

Section 4.10 - Process E10 - Receive Stimuli

Process Summary The Receive Stimuli process is concerned with the receipt and filing of all externally received stimuli. The stimuli are filed in a manner which makes the information easily accessible by the entire staff. The process is illustrated in Figure 4.10.1, and is also described in greater detail below.

Figure 4.10.1 Receive Stimuli Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E10: Receive Stimuli
I1: E1 Complete
S1: Project Charter
S2: Documents/ Working Papers
S3: Informal Communication
S4: Project Review Minutes
S5: Resource Allocation
S6: Master Project Schedule
Engineering Tasks for Process E10
No
Completion Conditions Achieved ?
Yes
I7: E10 Complete

Process Completed By Specification, Development and Certification teams.

Previous Process Project Information Management (E3)

Pre-condition Stimuli Received? (C5) = TRUE

Subsequent Process Project Information Management (E3)

Stimuli E1 Complete (I1), Project Charter (S1), Documents/Working Papers (S2), Informal Communication (S3), Project Review Minutes (S4), Resource Allocation (S5), Master Project Schedule (S6).

Responses E10 Complete (I7)

State Data Changes This process leads to new and modified material appearing in the Project Document Files (additional reference material) and the Project Management Files (additional management information). Unresolved Questions or Issues are modified when the stimuli received is the resolution to an external question or issue. The Project Management Files, specifically the project schedule, is used to determine a resource and effort allocation for the process.

Process Description Complete (achieve Completion Conditions for) the following sequence of engineering tasks for the Receive Stimuli process each time the process is invoked by condition C5 being true:

1. Receive the stimuli.
2. If the stimulus is S3 (Informal Communication), then it must be written down if it is to be archived.
3. The information must then be inserted into the state data, in the manner described below:
   a) S1, S5 and S6 (Project Charter, Resource Allocation and Master Project Schedule) are put in the Project Management Files.
   b) S2, S3, and S4 (Documents/Working Papers, Informal Communication, and Project Review Minutes) are put into the Project Document File.
4. Documents/Working Papers include information that the engineering team requires. This information includes requirements, models, external trade studies, constraints, interfaces, etc.
5. Notify the rest of the engineering team of the stimuli received.
6. If any of the information received is the answer to an Unresolved Question or Issue, then that section of state data must also be modified in order to reflect the resolution. In that way, the resolution of the question can be kept with the question itself.

Measurement Data Generated Effort

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

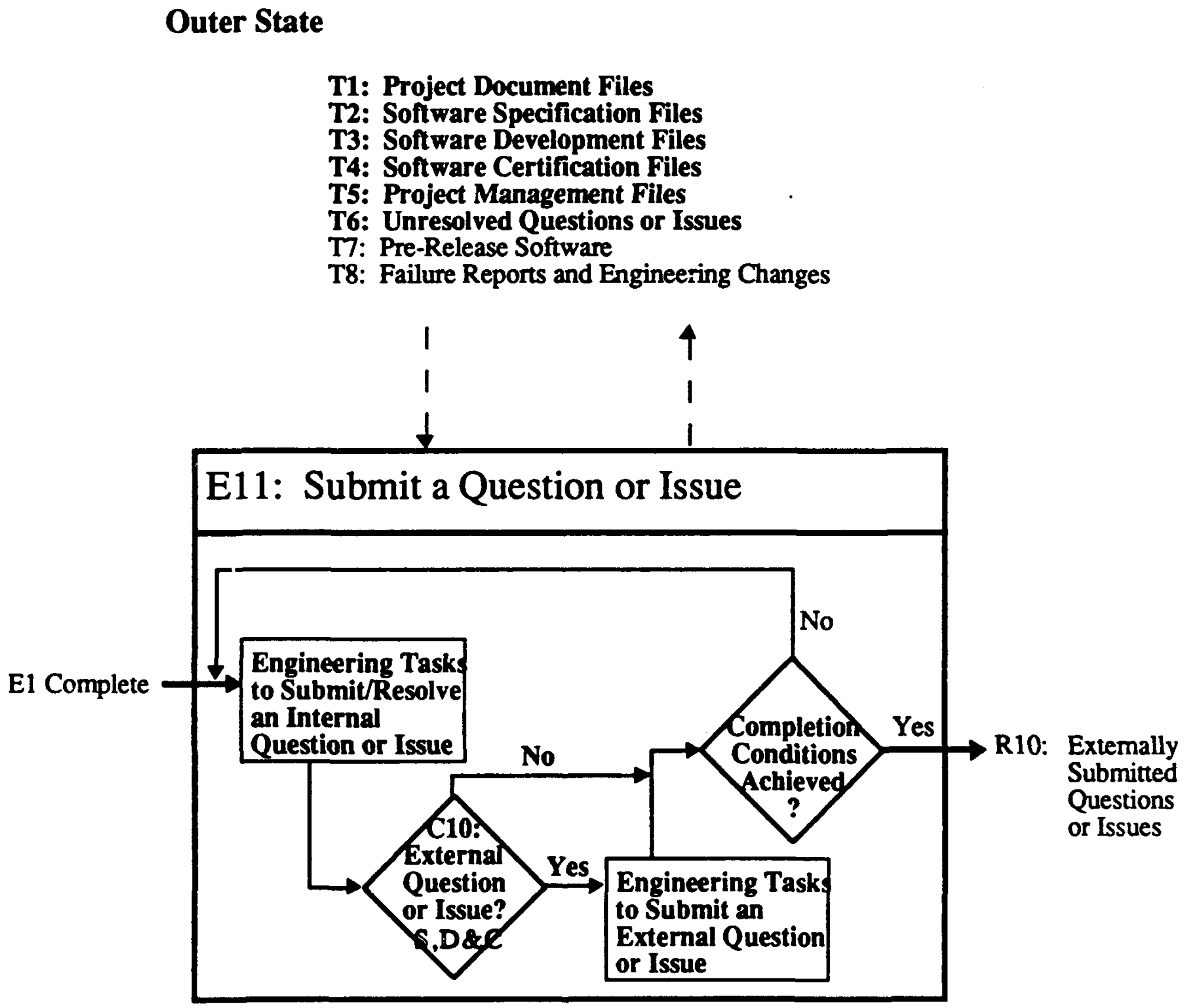
1. Have all stimuli been moved to all of the correct state data?
2. Have all other additions to, deletions from and modifications of the state data been completed?
3. Have all necessary measures been gathered?
4. Have all engineering team members who need the information received been notified?

Keyword References None

Section 4.11 - Process E11 - Submit a Question or Issue

Process Summary The Submit a Question or Issue process is concerned with questions or issues which are to be submitted or resolved internally and/or submitted externally. This is also where the responses to internal questions will appear. The process is illustrated in Figure 4.11.1, and is also described in greater detail below.

Figure 4.11.1 Submit a Question or Issue Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E11: Submit a Question or Issue
E1 Complete
Engineering Tasks to Submit/Resolve an Internal Question or Issue
No
No
C10: External Question or Issue? S,D&C
Yes
Engineering Tasks to Submit an External Question or Issue
Completion Conditions Achieved ?
Yes
R10: Externally Submitted Questions or Issues

Process Completed By Specification, Development and Certification teams.

Previous Process Project Information Management (E3)

Pre-condition Questions or Issues? (C6) = TRUE

Subsequent Process Project Information Management (E3)

Stimuli E1 Complete (I1)

Responses Externally Submitted Questions or Issues (R10)

State Data Changes This process leads to new and modified material appearing in Unresolved Questions and Issues (resolutions to existing questions or issues). All other state data is used as reference in order to help resolve internal questions or issues. The Project Management Files, specifically the project schedule, is used to determine a resource and effort allocation for the process.

Process Description Questions or issues are first submitted internally. The intent is to resolve as many questions/issues as possible internally. If the question or issue cannot be resolved internally, then it is submitted externally to be resolved.

The following algorithm is used to complete the Submit a Question or Issue process, until the Completion Conditions are achieved:

```
con
    Submit an internal question or issue. The question will be placed in state data.
    Resolve any submitted internal questions or issues by analyzing available material in order
    to reach a resolution. The resolution will then be placed in state data along with any other
    material generated.
noc
if C10: External Question or Issue
    then
        Submit an Externally Submitted Question or Issue (R10) for resolution
fi
```

C10: External Question or Issue is true when there is a question or issue that cannot be solved internally and must be solved externally. Usually this occurs after an attempt has been made to resolve the question or issue internally. The only option is to submit the question or issue externally.

Measurement Data Generated Effort, State Data Produced

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

1. Have all additions to, deletions from and modifications of the state data been made?
2. Have all questions that cannot be resolved by the engineering team been formally submitted as responses?
3. Have all necessary measures been made?

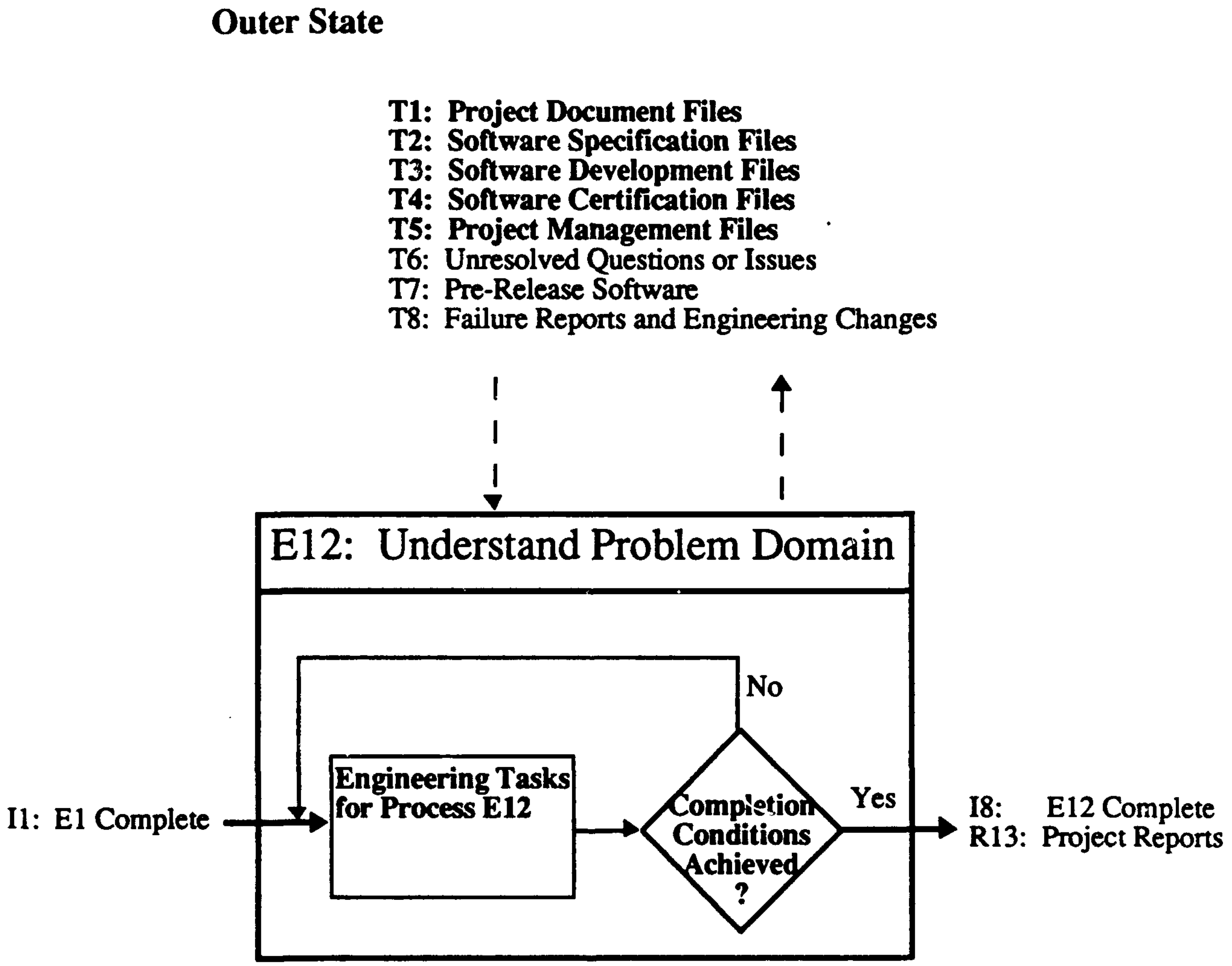
Keyword References -

Unresolved Questions or Issues - Section 5.6

Section 4.12 - Process E12 - Understand Problem Domain

Process Summary The Understand Problem Domain process leads the engineering team to investigate, analyze and document the problem to be solved in order to understand it better. This understanding can be assisted by reading, interviews, modeling, trade studies, prototyping and focus group interviews. The process is illustrated in Figure 4.12.1, and is also described in greater detail below.

Figure 4.12.1 Understand Problem Domain Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E12: Understand Problem Domain
No
I1: E1 Complete
Engineering Tasks for Process E12
Completion Conditions Achieved ?
Yes
I8: E12 Complete
R13: Project Reports

Process Completed By Specification, Development and Certification teams.

Previous Processes Software Solution Specification (E4)

Pre-condition None for the first iteration, Software Ready For Release? (C1) = FALSE for subsequent iterations.

Subsequent Process Write Construction Plan (E15)

Stimuli E1 Complete (I1)

Responses E12 Complete (I8), Project Reports (R13)

State Data Usage This process directly inserts information into the first four engineering files, as more is understood about the project. Existing information may also be modified as the level of understanding increases. The project schedule in the Project Management Files is used to determine the effort and resource allocation for the process.

Process Description Complete the following engineering tasks during the Understand Problem Domain process. These engineering tasks can be conducted concurrently until the completion conditions are achieved.

1. Perform sufficient iterations of investigation, analysis and documentation to achieve the passage criteria. Investigations may include: interviews, top down exploration, descriptive modeling (data flow diagrams, state transition diagrams, entity relationship diagrams, clear boxes,...), usage scenario creation, trade studies, prototyping and focus group interviews.
2. Read all available literature that may clarify problem domain issues.
3. Define requirements, constraints and objectives, by modifying system and software requirements submitted by external organizations, and by creating new ones.
4. Use top-down and bottom-up thinking to understand the problem domain.
5. If any acquired knowledge is considered to have value outside of the project team, then a Project Report that contains the information is submitted.

Measurement Data Generated Effort

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

1. Has the project manager determined that a sufficient understanding of the problem domain has been gained to complete this process?
2. Has the Specification team developed an approved copy of the specifications?
3. Is there a clear definition of what is to be accomplished by the software?
4. Has a clear description of all external interfaces for the software (converting all real-world stimuli into digital stimuli and digital responses into real-world responses) been determined?
5. Is there a clear model which describes the problem domain?
6. Has the model been judged as adequate to describe the problem domain?
7. Do requirements express conceptual integrity (including derived requirements generated as well as external requirements)?
8. Have all additions to, deletions from and modifications of the state data been made?
9. Have all necessary measures been gathered?
10. Have all external responses been submitted?

11. Have all pertinent reviews for this process been completed (for example, if this process is following 2167A, the System Requirements Review and the System Design Review need to have been completed)?
12. Have all action items generated during reviews that pertain to this process been completed?

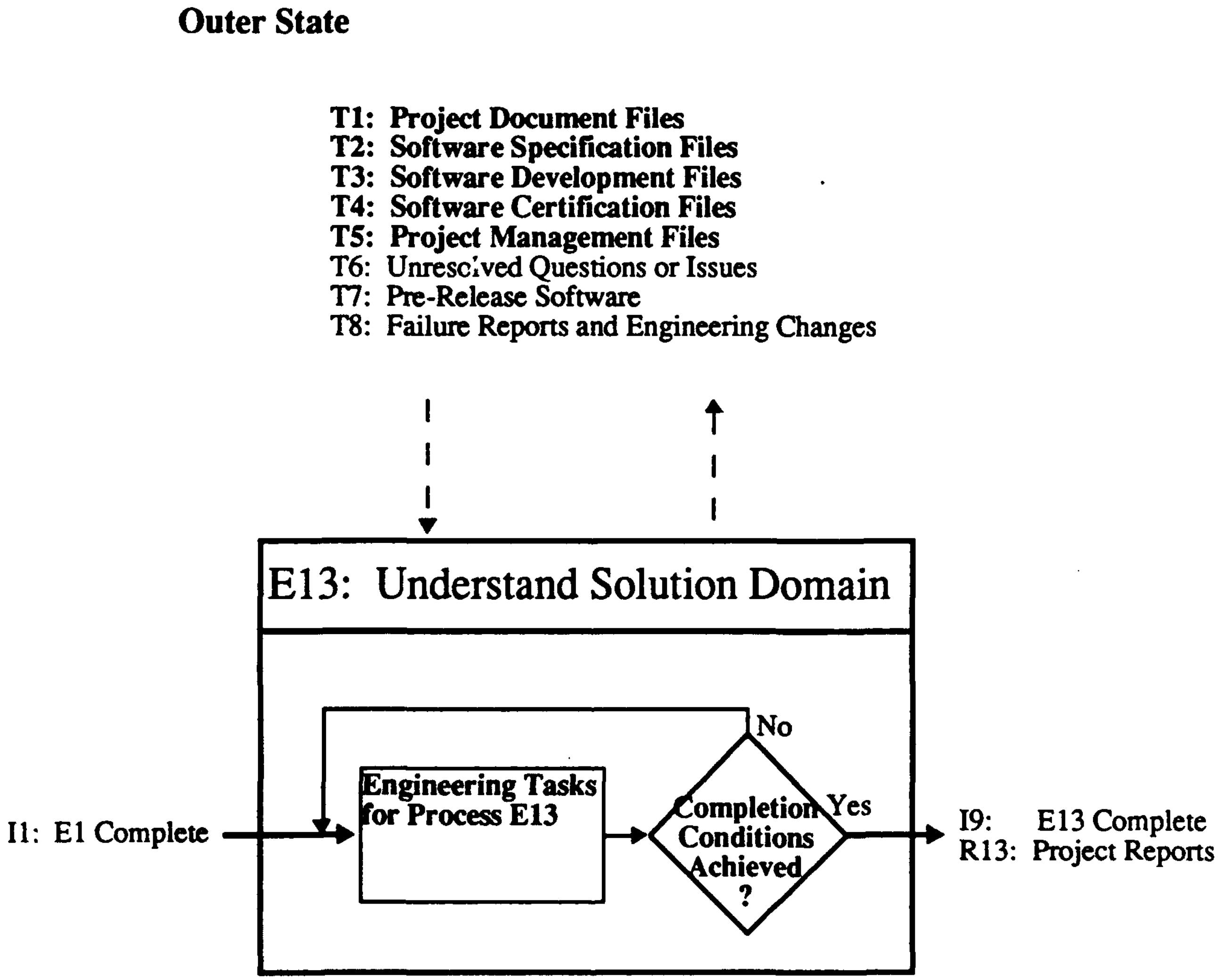
Keyword References

Review - Section 4.8

Section 4.13 - Process E13 - Understand Solution Domain

Process Summary The Understand Solution Domain process leads the engineering team to investigate, analyze and document the solution to the problem in order to understand it better. This understanding can be assisted by reading, interviews, trade studies, prototyping and focus group interviews. This process is also the time to analyze opportunities for reuse of specifications and other material from previous projects or repository libraries. The process is illustrated in Figure 4.13.1, and is also described in greater detail below.

Figure 4.13.1 Understand Solution Domain Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E13: Understand Solution Domain
I1: E1 Complete
Engineering Tasks for Process E13
Completion Conditions Achieved ?
No
Yes
I9: E13 Complete
R13: Project Reports

Process Completed By Specification, Development and Certification teams.

Previous Processes Software Solution Specification (E4)

Pre-condition None for the first iteration, Software Ready For Release? (C1) = FALSE for subsequent iterations.

Subsequent Process Write Construction Plan (F15)

Stimuli E1 Complete (I1)

Responses E13 Complete (I9), Project Reports (R13)

State Data Usage This process directly inserts information into the first four engineering files, as more is understood about the project. Existing information may also be modified as the level of understanding increases. The project schedule in the Project Management Files is used to determine an effort and resource allocation for this process.

Process Description Complete the following engineering tasks during the Understand Solution Domain process. These engineering tasks can be conducted concurrently until the Completion Conditions are achieved.

1. Perform sufficient iterations of investigation, analysis and documentation to achieve the Completion Conditions. Investigations may include: prototyping, focus group interviews, bottom up exploration, analytic modeling, and trade studies.
2. Read all available literature that may clarify solution domain issues.
3. Examine existing solutions to related problems (typically in other organizations).
4. Use top-down and bottom-up thinking in order to understand the solution domain.
5. Examine opportunities for reuse of specifications and previously used development, certification, and management file material.
6. Develop a philosophy for implementing the solution.
7. If any acquired knowledge is considered to have value outside of the project team, then a Project Report that contains the information is issued.

Measurement Data Generated Effort

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

1. Has the project manager determined that a sufficient understanding of the solution domain has been gained to complete this process?
2. Has the Specification team developed an approved copy of the specifications?
3. Is there a clear definition of what is to be accomplished by the software?
4. Has a clear description of all external interfaces for the software (converting all real-world stimuli into digital stimuli and digital responses into real-world responses) been determined?
5. Is there a clear model which describes the solution domain?
6. Has the model (programming philosophy) been judged as adequate to describe the solution domain?
7. Have all pertinent reviews for this process been completed (for example, if this process is following 2167A, the System Requirements Review and the System Design Review need to have been completed)?

8. Have all additions to, deletions from and modifications of the state data been made?
9. Have all necessary measures been gathered?
10. Have all external responses been submitted?
11. Have all action items generated during reviews that pertain to this process been completed?

Keyword References

Review - Section 4.8

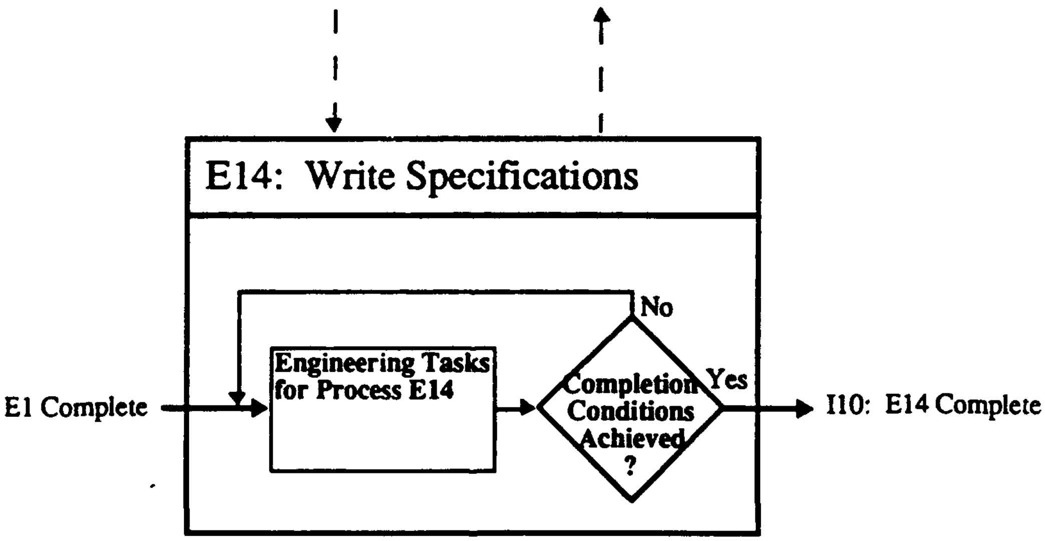
Section 4.14 - Process E14 - Write Specifications

Process Summary The Write Specifications process is where the Specification team writes the first five volumes of the six volume Specifications. Those five volumes are: the Mission Volume, the User's Reference Manual Volume, the Functional (Black Box) Specification Volume, the Functional Specification Verification Volume, and the Usage Profile Volume. The process is illustrated in Figure 4.14.1, ar 1 is also described in greater detail below.

Figure 4.14.1 Write Specifications Process

Outer State

T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes


E14: Write Specifications
No
Engineering Tasks for Process E14
Completion Conditions Achieved ?
Yes
E1 Complete
I10: E14 Complete

Process Completed By Specification team

Previous Processes Software Solution Specification (E4)

Pre-condition None for the first iteration, Software Ready For Release? (C1) = FALSE for subsequent iterations.

Subsequent Process Write Construction Plan (E15)

Stimuli E1 Complete (I1)

Responses E14 Complete (I10)

State Data Usage This process creates the Specifications in the Software Specification Files, so that data is affected. Additionally, Project Document Files and Project Management Files are used as reference material in writing the first five volumes of the Specifications. The project schedule of the Project Management Files is used in order to determine effort and resource allocation for the process.

Process Description Complete (achieve Completion Conditions for) the following engineering tasks during the Write Specifications process:

1. Use the knowledge gained from understanding the problem and solution domains to write the Specifications. Write the Specifications in the required format. Specifically, the specifications consist of six documents: the Mission Volume, the Functional (Black Box) Specification Volume, the Functional Specification Verification Volume, the User's Reference Manual Volume, the Usage Profile Volume, and the Construction Plan. The first five documents must be written during this stage, using a spiral workplan approach.
2. Use bottom-up exploration to better understand the solution domain in order to create the Specifications in a top-down manner.
3. Determine if it is possible to reuse Specifications from other projects. If so, use the material.
4. The Function Specification Verification Volume must be emphasized. The is the volume where the Specification team must persuade all readers that the Functional Specification is correct.
5. Once the Specification team believes that they have written a 'complete' version of the Specifications, the Development and Certification teams may need to review and approve the Specifications in order to ensure that both teams thoroughly understand the material presented. If material is reviewed and corrections are needed, then the Specifications must be corrected and re-reviewed by the Development and Certification teams.
6. Once the Specification team is convinced that this task is completed, the Specifications must be reviewed and approved by management as a formal review.
7. If the Specifications are not approved, the Specification team continues to refine the Specifications, otherwise, this process is completed.
8. Once this process is considered complete, the Specifications must be put under some configuration management to control change to the document. Specifically, only the Specification team should be able to modify the Specifications.

Measurement Data Generated Effort, State Data Produced

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

1. Are the Specifications complete, consistent and unambiguous to the extent possible?
2. Have the Specifications been reviewed and accepted by the Certification and Development teams?
3. Has the Specification team demonstrated that the Specifications describe a cost effective solution?

4. Is the present version of the Specifications consistent with the Master Project Schedule, Project Charter, etc.?
5. Have the Specifications been placed under configuration management?
6. Are the Specifications consistent with the system requirements, and any software requirements that were submitted by management.
7. Are the Specifications for the software consistent with the rest of the system, including hardware and human activities (for example, in terms of 2167A, these issues would need to be resolved by the System/Segment Design Document)?
8. Are the Specifications consistent with the software development plan?
9. Have the five volumes of the Specifications been placed in the correct state data?
10. Have all necessary measures been gathered?
11. Have all pertinent reviews for this process been completed?
12. Have all action items generated during reviews that pertain to this process been completed?
13. Havr all information to be preserved been placed in the correct state data?

Keyword Reference

Review - Section 4.8

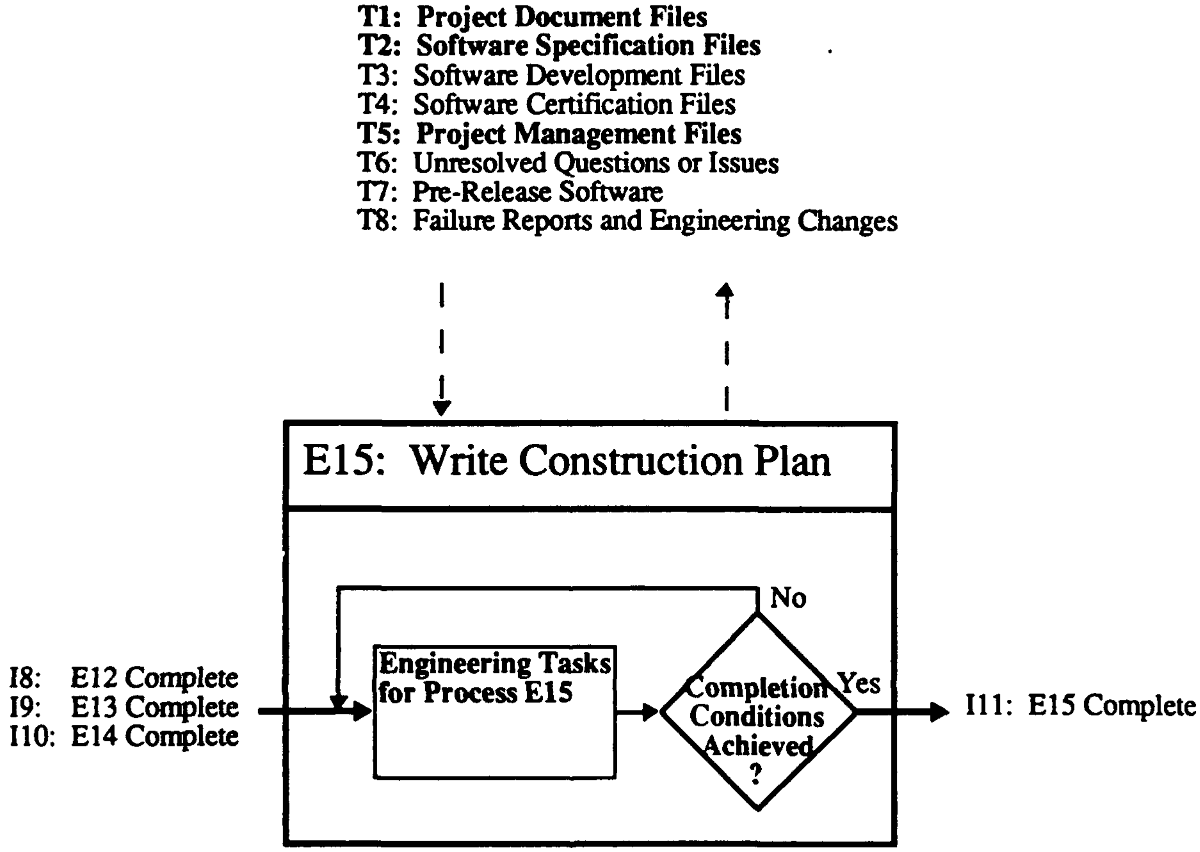
Specification, Mission Volume, Functional (Black Box) Specification Volume, Functional Specification Verification Volume, User's Reference Manual Volume, Usage Profile Volume, Construction Plan - Section 5.2 and Section 8

Section 4.15 - Process E15 - Write Construction Plan

Process Summary The Write Construction Plan process is where the Specification team writes the sixth volume of the Specifications, the Construction Plan. The Specification team must write the plan to ensure that increments are developed in the most efficient manner possible to permit usage testing and to permit the incorporation of reuse components. The construction planning phase is the time to plan for reliability by selecting the construction plan that will yield the greatest reliability. The process is illustrated in Figure 4.15.1, and is also described in greater detail below.

Figure 4.15.1 Write Construction Plan Process

Outer State


T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E15: Write Construction Plan
No
Engineering Tasks for Process E15
Completion Conditions Achieved ?
Yes
I8: E12 Complete
I9: E13 Complete
I10: E14 Complete
I11: E15 Complete

Process Completed By Specification team

Previous Processes Understand Problem Domain (E12), Understand Solution Domain (E13), and Write Specifications (E14)

Pre-condition Continue with Project (C7) = TRUE on subsequent iterations

Subsequent Process Software Solution Specification (E4)

Stimuli E12 Complete (I8), E13 Complete (I9), E14 Complete (I10)

Responses E15 Complete (I11)

State Data Usage This process, of course, creates the sixth volume of the Specifications in the Software Specification Files, so that data is affected. Additionally, the Project Document Files and the Project Management Files are used as reference material in writing the this section of the Specifications. The project schedule in the Project Management Files is used to determine effort and resource allocations for this process.

Process Description Complete (achieve Completion Conditions for) the following engineering tasks during the Write Construction Plan process:

1. Write the Construction Plan, which is the sixth volume of the Specifications, to ensure that the increments will be developed in the most efficient manner possible to permit usage testing. The plan should allow for the Specification, Development and Certification teams to work in parallel.
2. Plan for reuse and reliability. See the IR-70 Phase 1 report "STARS-Cleanroom Reliability: Cleanroom Ideas In the STARS Environment," September 30, 1989.
3. When the project is replanned, the Construction Plan must be rewritten.
4. The Construction Plan is preserved by the engineering team as the sixth volume of the Specifications.
5. Creation of the Construction will lead to the Project Management Files being modified (by process E9), as now the increments are placed in a schedule, which leads to task assignments being generated. The Construction Plan determines what units, functions, etc. will be in each increment. Each will be made into sets of task assignments.

Measurement Data Generated Effort, State Data Produced

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

1. Will the Construction Plan meet the following constraints:
   a) resources,
   b) budget,
   c) schedule,
   d) provides the level of quality desired for the project?
2. Has the Construction Plan decomposed the project into the best set of executable increments from the following standpoints:
   a) Size suitable to development teams' capabilities (all modules should normally be under 10 KLOC barring exceptional circumstances),
   b) Complexity suitable to certification team's capabilities,
   c) Reuse,

   d) Prototyping,
   e) Complexity/size of the predicted solution domain,
   f) Hardware domain/operating environment of the system to be implemented,
   g) Client/deliverable obligations?

3. Are the Specifications consistent with the software development plan?
4. Has the Construction Plan been placed in the correct state data?
5. Have all necessary measures been gathered?
6. Have all pertinent reviews for this process been completed?
7. Have all action items generated during reviews that pertain to this process been completed?
8. Have all information to be preserved been placed in the correct state data?

Keyword References

Review - Section 4.8
Specifications, Construction Plan - Section 5.2 and Section 8

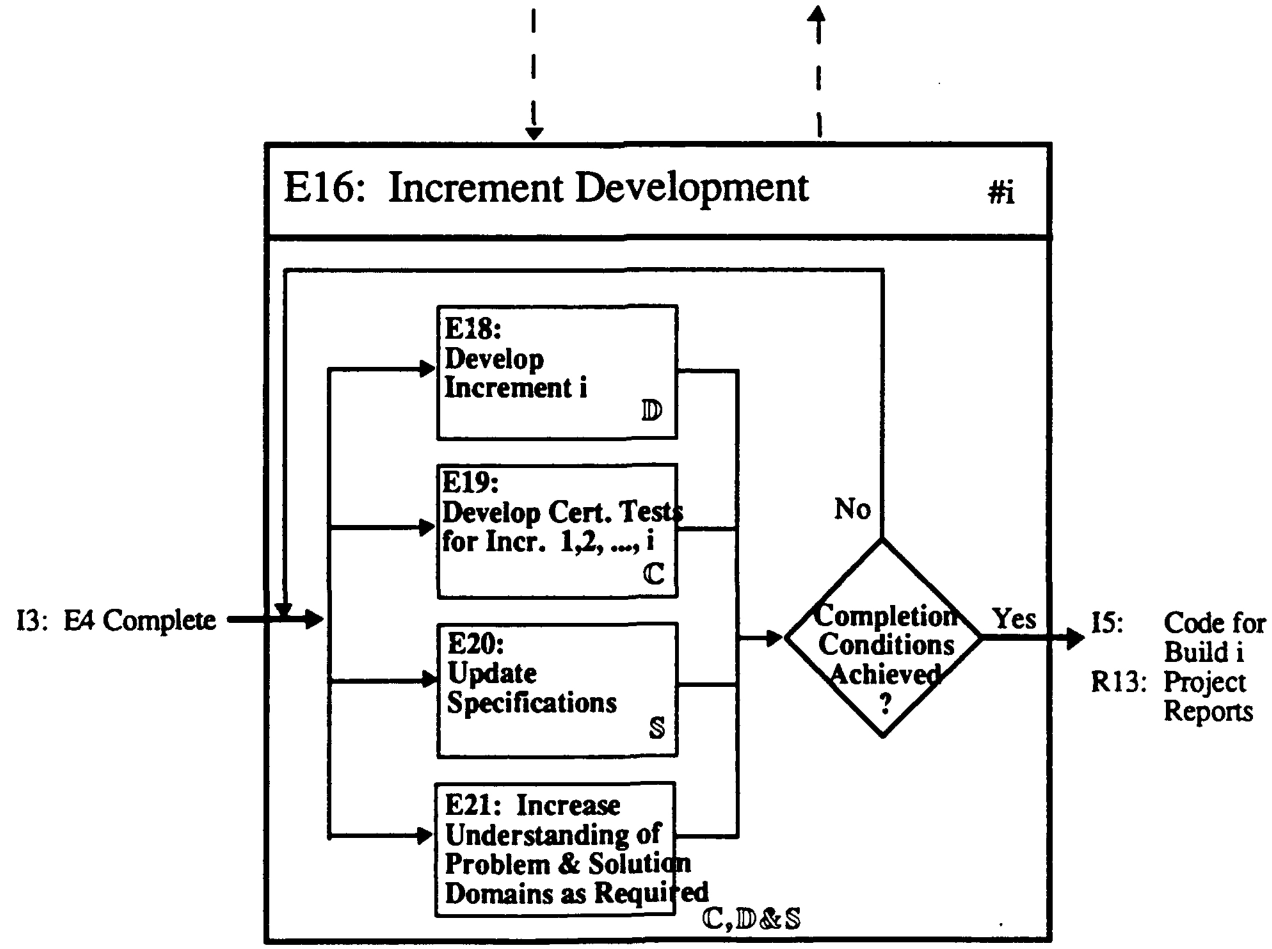
Section 4.16 - Process E16 - Increment Development

Process Summary The Increment Development process conducts all of the processes that are a part of developing an increment. These processes include the development and verification of the code for an increment, the development of certification tests for an increment, updating the specifications, and increasing the understanding of the problem and solution domains. The process is illustrated in Figure 4.16.1, and is also described in greater detail below.

Figure 4.16.1 Increment Development Process

Outer State

T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes


E16: Increment Development
#i
E18: Develop Increment i
D
E19: Develop Cert. Tests for Incr. 1,2, ..., i
C
E20: Update Specifications
S
E21: Increase Understanding of Problem & Solution Domains as Required
C,D&S
No
Completion Conditions Achieved ?
Yes
I3: E4 Complete
I5: Code for Build i
R13: Project Reports

Process Completed By Specification, Development and Certification

Previous Processes Software Development and Certification (E5)

Pre-condition None

Subsequent Process Software Development and Certification (E5)

Stimuli E4 Complete (I3)

Responses Code for Build i (I5), Project Reports (R13)

State Data Usage This process does not modify state data, although its sub-processes do. The effect of the sub-processes on the state data will be detailed in each of the sub-process descriptions.

Measurement Data Generated Effort

Process Description Follow the algorithm below for the Increment Development process:

```
do
    con
        use E18:  Develop Increment i
        use E19:  Develop Certification Tests for Increments 1..i.
        use E20:  Update Specifications
        use E21:  Increase Understanding of Problem and Solution Domains as Required
    noc
until
    Completion Conditions achieved
od
```

E18: Develop Increment 1
E19: Develop Certification Plan and Tests for Increments 1...i
E20: Update Specifications
E21: Increase Understanding of Problem and Solution Domain as Required
are described in Sections 4.18, 4.19, 4.20 and 4.21, respectively.

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

1. Have all necessary measures been gathered?
2. Have all completion conditions for all four sub-processes (E18, E19, E20, E21) been achieved?

Keyword Reference None

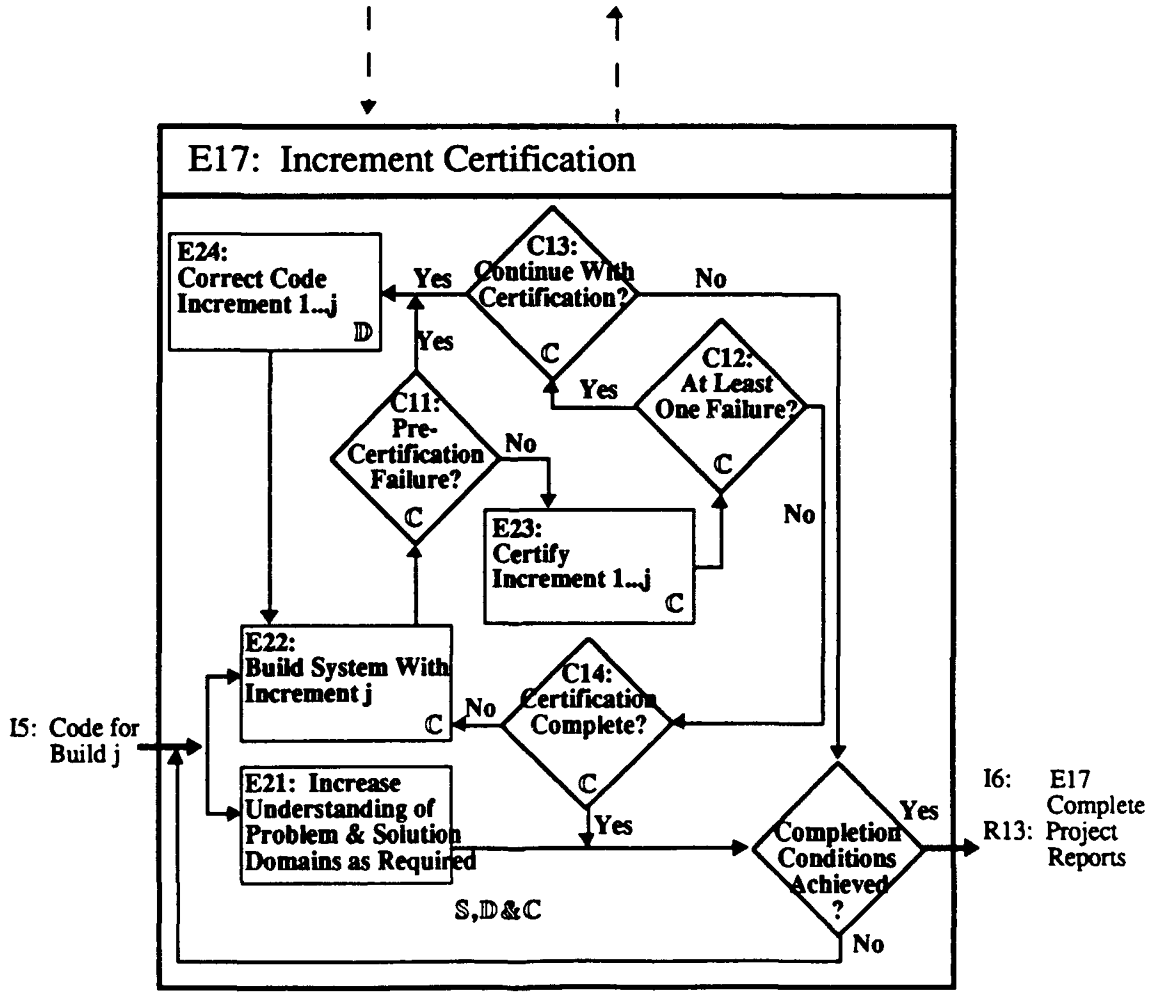
Section 4.17 - Process E17 - Increment Certification

Process Summary The Increment Certification process conducts all of the processes that are a part of certifying an increment. Code is put under configuration control, compiled, assembled and certified. If failures are found, the code is corrected and the process continues. This continues until certification for an increment is complete or the effort is abandoned since it is determined that the software exhibits too many failures. Additionally, the engineering team may work to increase their understanding of the problem and solution domains. The process is illustrated in Figure 4.17.1, and is also described in greater detail below.

Figure 4.17.1 Increment Certification Process

Outer State

T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes


E17: Increment Certification
E24: Correct Code Increment 1...j
D
Yes
C13: Continue With Certification?
C
No
Yes
Yes
C12: At Least One Failure?
C
C11: Pre-Certification Failure?
C
No
E23: Certify Increment 1...j
C
No
E22: Build System With Increment j
C
No
C14: Certification Complete?
C
Yes
I5: Code for Build j
E21: Increase Understanding of Problem & Solution Domains as Required
S,D&C
Completion Conditions Achieved ?
Yes
No
I6: E17 Complete
R13: Project Reports

Process Completed By Specification, Development and Certification teams.

Previous Processes Software Development and Certification (E5)

Pre-condition Next Increment In Construction Plan Ready? (C8) = TRUE and Final Increment? (C9) = FALSE

Subsequent Process Software Development and Certification (E5)

Stimuli Code for Build j (I5)

Responses E17 Complete (I6), Project Reports (R13)

State Data Usage This process does not modify state data, although its sub-processes do. The effect of the sub-processes on the state data will be detailed in each of the sub-process descriptions.

Process Description Follow the algorithm appearing below to complete the Increment Certification process:

```
do
    con
        use E21: Increase Understanding of Problem and Solution Domains as Required
        do
            use E22: Build System with Increment j
            if C11: Pre-Certification Failure?
                then
                    use E24: Correct Code Increment 1..j
                else
                    use E23: Certify Increments 1..j
                    if C12: At Least One Failure?
                        then
                            if C13: Continue with Certification?
                                then
                                    use E24: Correct Code Increment 1..j
                                else
                                    Exit process
                            fi
                    fi
            fi
        until
                C14: Certification Complete?
        od
    noc
until
    Completion Criteria achieved
od
```

C11: Pre-Certification Failure is true if a failure has been observed while compiling or assembling the system.

C12: At Least One Failure is true if failures have been observed while executing certification tests.

C13: Continue with Certification is true if the Certification team has decided to halt certification until the Development team corrects the observed failures. This includes comparing present with predicted reliability for the increment. C13 is false if the Certification team has determined, as a result of the volume or severity of failures observed, that further certification is not beneficial.

C14: Certification Complete is true when the Certification team determines that certification is complete for the entire system. If C14 is false, certification continues with the next increment.

E21: Increase Understanding of Problem and Solution Domain as Required
E22: Build System with Increment j
E23: Certify Increments 1...j
E24: Correct Code Increment 1...j
are described in Sections 4.21, 4.22, 4.23 and 4.24, respectively

Measurement Data Generated Effort

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

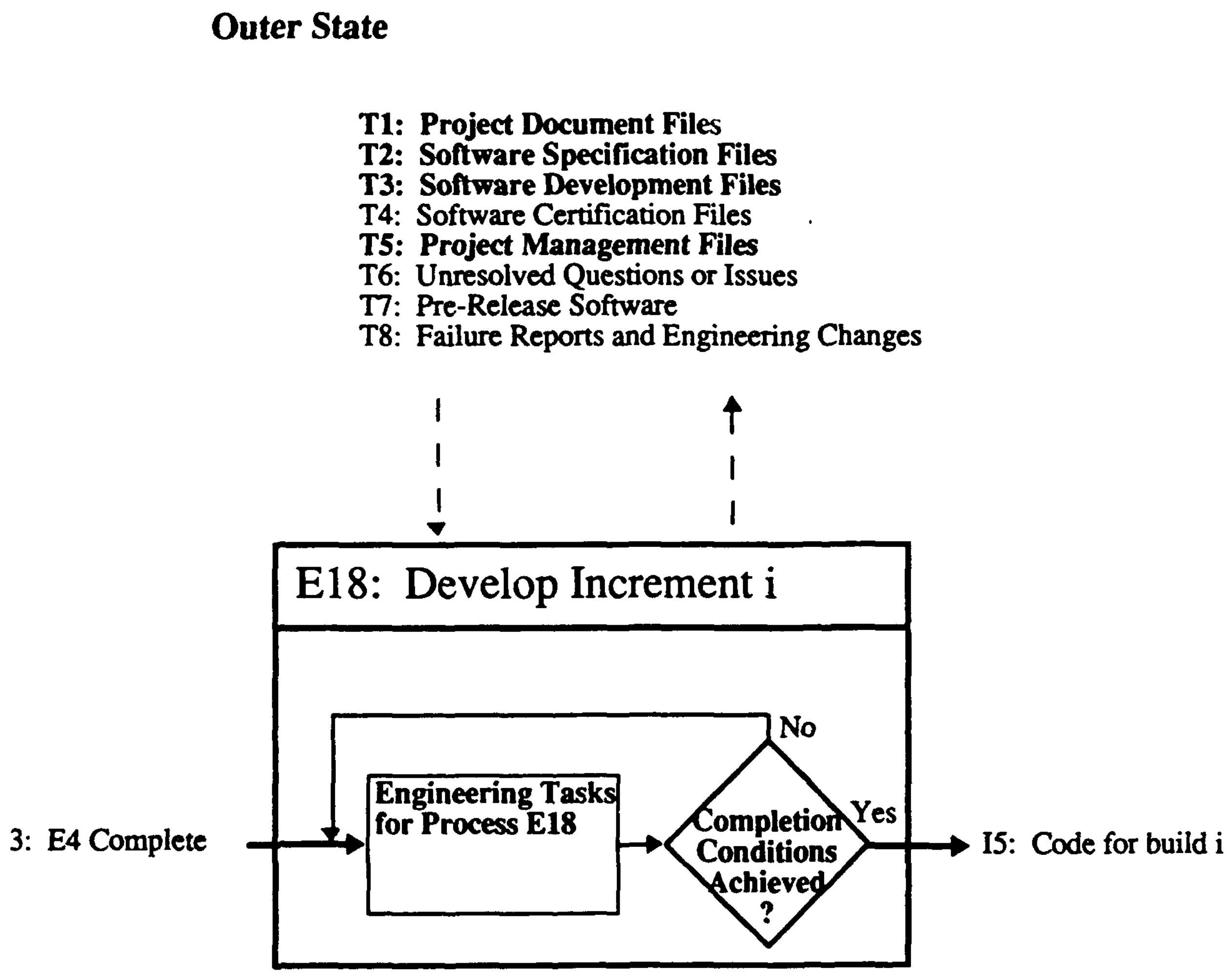
1. Have all necessary measures been gathered?
2. Have all completion conditions for all four sub-processes (E21, E22, E23, E24) been achieved?

Keyword Reference None

Section 4.18 - Process E18 - Develop Increment i

Process Summary The Develop Increment i process conducts the design and implementation of the software product. The Development team follows the Box Structure Algorithm for design and uses Stepwise Refinement with Functional Verification for implementation. The process is illustrated in Figure 4.18.1, and is also described in greater detail below.

Figure 4.18.1 Develop Increment i Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E18: Develop Increment i
3: E4 Complete
Engineering Tasks for Process E18
No
Completion Conditions Achieved ?
Yes
I5: Code for build i

Process Completed By Development team

Previous Process Increment Development (E16)

Pre-condition None

Subsequent Process Increment Certification (E17)

Stimuli E4 Complete (I3)

Responses Code for build i (I5)

State Data Usage This process involves the design and code of the software product, which is the primary content of the Software Development Files. That material is created and modified by this process. The Project Document Files, Software Specification Files and Project Management Files are primarily used as reference sources. The project schedule in the Project Management Files is used to determine effort and resource allocation for the process.

Process Description Starting with the Specifications and the Software Development and Project Document Files as primary references, develop code using the following sequence of engineering tasks. Continue until the Completion Conditions are achieved.

1. Define Black Box by
   1.1 Defining stimuli
   1.2 Defining responses in terms of stimuli histories
   1.3 Validating black box

2. Define State Box by
   2.1 Defining state data to represent stimuli histories
   2.2 Selecting state data to be maintained at this level
   2.3 Modifying black box to represent repositories in terms of stimuli and the state data being maintained at this level
   2.4 Recording the type of references to state data by stimuli
   2.5 Verifying the state box

3. Define Clear Box by
   3.1 Defining data abstractions for each state data
   3.2 Modifying the state box to represent responses in terms of stimuli, this level's state data and invocations of lower level black boxes
   3.3 Verifying the clear box

4. Repeat steps 1 - 3 until a sufficiently detailed level of clear boxes exist to permit coding to begin.

5. Develop code by
   5.1 Writing code for each clear box using stepwise refinement
   5.2 Individually verifying the correctness of code written by using functional verification
   5.3 Verifying the correctness of the increment by conducting peer reviews.

6. Submit code to Certification team as the response from this process.

Remember to examine opportunities for reuse of boxes and code from previous products. Look for common services and reused boxes and code inside this project.

Measurement Data Generated Effort State Data Produced

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process:

Black, State and Clear boxes and code

1. Have team reviews been held for all boxes, trade studies and code?
2. Have all boxes, trade studies and code passed their final review?
3. Have all project/installation standards for design languages and code been adhered to, including:
   a) Using only the four basic structures?
   b) Following language syntax conventions?
   c) Providing function commentary where needed?
4. Have all pertinent reviews for this process been completed?
5. Have all action items generated during reviews that pertain to this process been completed?
6. Are the items developed consistent with the software development plan?

Black boxes

7. Have all black box functions been clearly defined in terms of stimulus histories, and have been defined using an acceptable format?
8. Have all black box functions been verified to their specification (Note: A higher level clear box is a lower level black box's specification)?
9. Does the black box function process all stimuli values, both valid and invalid?

State boxes

10. Have all state box functions been validated to be equivalent to their black box functions?
11. Does the state box function process all stimuli values, both valid and invalid leaving the state data in a valid state?
12. Have trade studies been conducted for all state data decisions:
    a) For elaboration at the selected level in the usage hierarchy?
    b) Does state data abstraction have right balance between execution speed, performance and verifiability?
13. Has transaction closure been obtained for all state boxes?

Clear boxes

14. Have all clear box functions been verified to be equivalent to their state box functions?
15. Are all interfaces in the usage hierarchy well defined?
16. Has transaction closure been obtained for all clear boxes?
17. For concurrent clear boxes, has an appropriate resolve function been defined?
18. Have all opportunities for common services been evaluated to determine proper modularization?

Code

19. Have all procedures in the increment been verified/code read by the team?
20. Has code for all procedures been developed by stepwise refinement?
21. Have all refinements been proven:
    a) Simple ones by direct assertion?
    b) Complicated ones by formal proof?

Overall

22. Is the increment complete according to the items listed in the Construction Plan (Volume 6 of the Specifications)?
23. Have all state data in the Software Development Files been correctly added, modified or deleted?
24. Have all necessary measures been gathered?
25. Have all responses been submitted?
26. Have all information to be preserved been placed in the correct state data?

Keyword References

Black Box, State Box, Clear Box and Box Structures Algorithm

- Sections 9.2, 9.3 and 9.6
- Principles of Information Systems Analysis and Design (Mills, Linger, Hevner)
- "Stepwise Refinement and Verification of Box Structured Systems," IEEE Computer, June 1988 (Mills)
- "Designing Software Systems with Box Structures," SET Course

Code, Verification and Stepwise Refinement -

- Sections 9.4, 9.5 and 9.6
- Structured Programming: Theory and Practice (Linger, Mills and Witt)
- "Implementing Software Systems with Stepwise Refinement and Functional Verification," SET Course

Review - Section 4.8

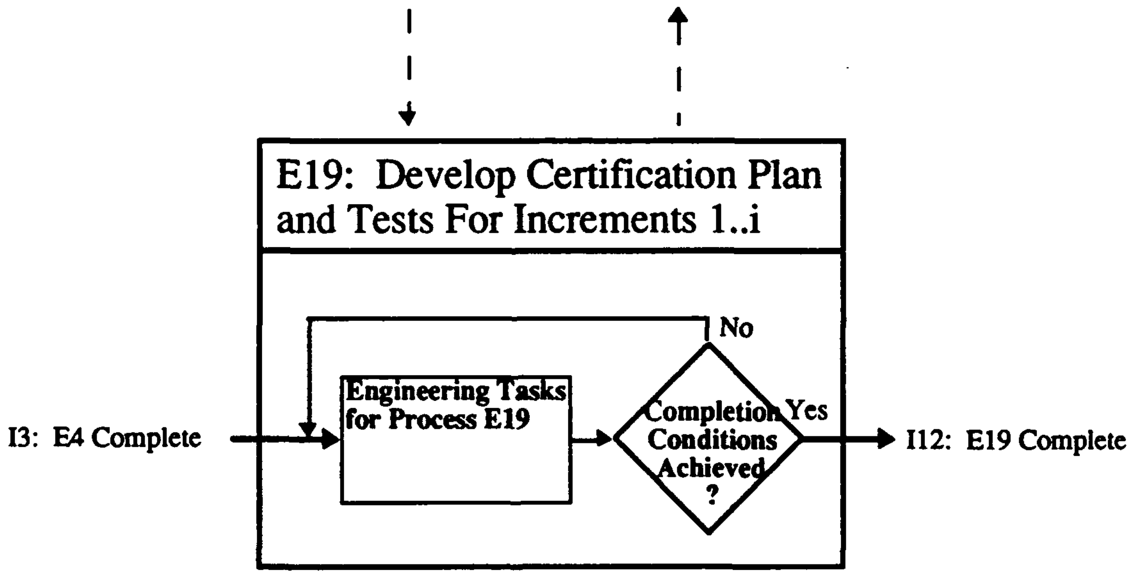
Section 4.19 - Process E19 - Develop Certification Plan and Tests For Increments 1..i

Process Summary The Develop Certification Tests For Increments 1..i process completes the preparation for the certification of the software product. A usage-based testing approach is developed, which will allow for usage testing to be conducted on the software. The process is illustrated in Figure 4.19.1, and is also described in greater detail below.

Figure 4.19.1 Develop Certification Plan and Tests for Increments 1...j Process

Outer State

T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes


E19: Develop Certification Plan and Tests For Increments 1..i
I3: E4 Complete
Engineering Tasks for Process E19
Completion Conditions Achieved ?
No
Yes
I12: E19 Complete

Process Completed By Certification team

Previous Processes Increment Development (E16)

Pre-condition None

Subsequent Process Increment Certification (E17)

Stimuli E4 Complete (I3)

Responses E19 Complete (I12)

State Data Usage This process involves the creation of certification tests for the software product, which is the primary content of the Software Certification Files. That material is created and modified by this process. The Project Document Files, Software Specification Files and Project Management Files are primarily used as reference sources. The project schedule in the Project Management Files is used to determine effort and resource allocations for this process.

Process Description The Software Specification Files contain the Usage Profile Volume, which presents, in the form of a matrix, the transition probabilities which define the probability of a user moving from any one program state to another program state. This matrix was developed by applying a Markov model to all the identified program states that the software can reach. Additionally, the Usage Profile Volume contains the stimuli to the system and the corresponding distributions for each stimulus in each state. Using the matrix and the stimuli information, the Certification team develops test cases by completing the following sequence of engineering tasks until the completion conditions are achieved:

1. Modify the usage profile according to the Construction Plan.
2. Determine the requirements coverage of each state and each state transition.
3. Determine how many test scenarios are required to test the increment to the desired level of reliability, given the expected rate of failures.
4. Specify the sampling scheme to be used to guide the testing for the increment. This includes the decision of whether both control flow and data will be randomized or just control flow.
5. Develop test scripts, or test script generators that list the stimuli and stimuli values which complete the program state transitions.
6. Develop test scenarios by creating random state transitions. Use the test scripts or script generators to generate each program state transition. This task may include the random generation of data.
7. Based on the test scenarios generated in task 6, compute expected outputs for each test case. The expected results form the basis of comparison for validation of test executions.
8. Set reliability targets and failure limits for the increment.

Measurement Data Generated Effort, State Data Produced

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process:

1. Do test scenarios reflect the operational profile of the software to be tested?
2. Can the results of executing all test scenarios be validated?
3. Have test passage criteria for the increment (such as number of test scenarios, failures or reliability goals) been determined and corresponding test information generated?
4. Have expected results been generated, and passage criteria determined for each test case?
5. Have sufficient test scenarios been generated to certify the increment to the desired level of reliability, given the expected rate of failure?
6. Is the increment complete according to the items listed in the Construction Plan (Volume 6 of the Specifications)?
7. Has all state data in the Software Certification Files been correctly added, changed or deleted?

8. Have all pertinent reviews for this process been completed?
9. Have all action items generated during reviews that pertain to this process been completed?
10. Have all information to be preserved been placed in the correct state data?

Keyword References

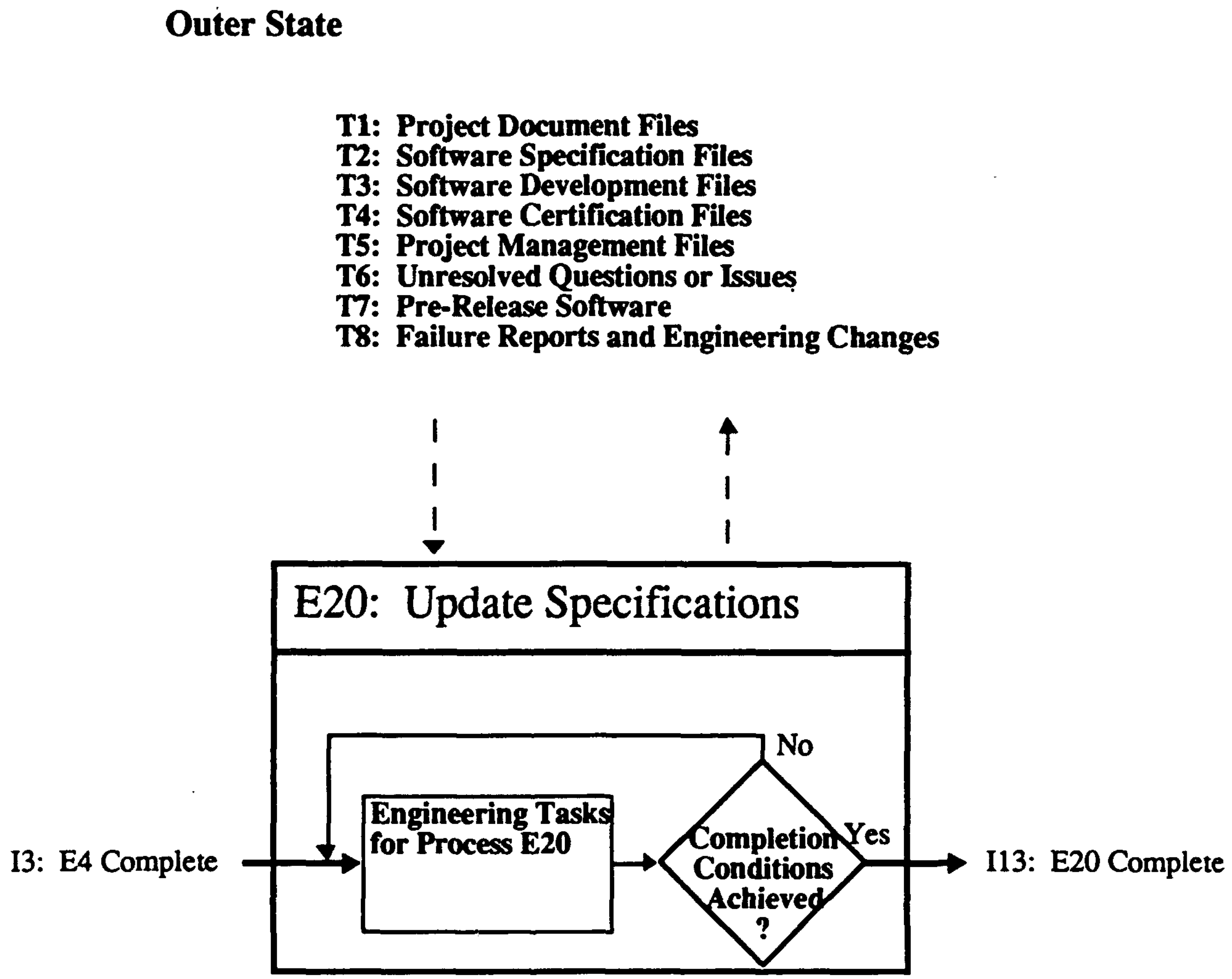
Test Script, Test Script Generator, Test Scenario

- Section 10
- "Engineering Software Under Statistical Quality Control," IEEE Software, November 1990 (Cobb, Mills)
- "Statistical Quality Control of Software System Development," SET Course

Section 4.20 - Process E20 - Update Specifications

Process Summary The Update Specifications process helps complete the Specifications, in the Software Specification Files, as needed. The process is illustrated in Figure 4.20.1, and is also described in greater detail below.

Figure 4.20.1 Update Specifications Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E20: Update Specifications
No
Engineering Tasks for Process E20
Completion Conditions Achieved ?
Yes
I3: E4 Complete
I13: E20 Complete

Process Completed By Specification team

Previous Process Increment Development (E16)

Pre-condition None

Subsequent Process Increment Certification (E17)

Stimuli E4 Complete (I3)

Responses E20 Complete (I13)

State Data Usage The Software Specification Files are modified during this process, as the intended software system is better understood. All other state data is used as reference material. The project schedule in the Project Management Files is used to determine resource and effort allocation for this process.

Process Description Complete (achieve Completion Conditions for) the following sequence of engineering tasks during the Update Specifications process:

1. Integrate any necessary modifications into the present version of the Software Specification Files, which are under configuration control. These modifications may be a result of errors, ambiguities, new requirements, problems encountered by the Development or Certifications teams in the course of their work, or descriptions for previously incomplete sections (e.g., TBD's). Publish a new version of the Specifications and distribute it to entire staff at the end of each increment.
2. All potential changes to the Software Specification Files must be approved and consistent with existing documentation. Use management assistance if necessary. Project manager must sign off on all specification changes.

Measurement Data Generated Effort, State Data Produced

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

1. Does the present version of the Specifications represent the precise system the Development team is implementing and the Certification team is certifying?
2. Is the present version of the Specifications consistent with the Master Project Schedule, Project Charter, etc.?
3. Is the present version of the Specifications consistent with the software development plan?
4. Are the Specifications for the software consistent with the rest of the system, including hardware and human activities (for example, in terms of 2167A, these issues would need to be resolved by the System/Segment Design Document)?
5. Have all pertinent reviews for this process been completed?
6. Have all action items generated during reviews that pertain to this process been completed?
7. Have all additions, changes or deletions from the Software Specification Files been correctly done?
8. Have all information to be preserved been placed in the correct state data?

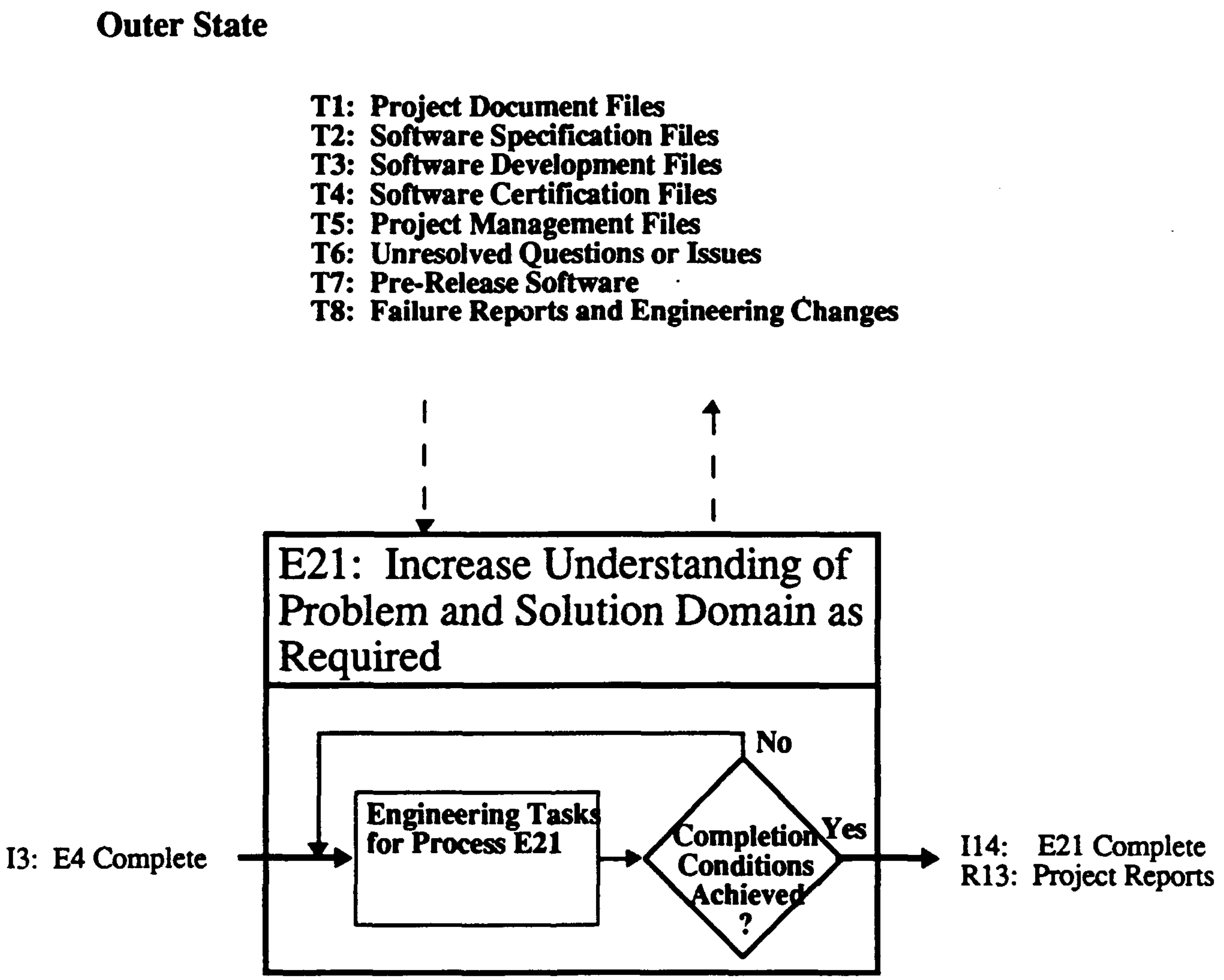
Keyword Reference

Review - Section 4.8
Section 8

Section 4.21 - Process E21 - Increase Understanding of Problem and Solution Domain as Required

Process Summary The Increase Understanding of Problem and Solution Domain as Required process is an opportunity for the engineering team to better understand the project. The process is illustrated in Figure 4.21.1, and is also described in greater detail below.

Figure 4.21.1 Increase Understanding of Problem and Solution Domain as Required Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E21: Increase Understanding of Problem and Solution Domain as Required
No
Engineering Tasks for Process E21
Completion Conditions Achieved ?
Yes
I3: E4 Complete
I14: E21 Complete
R13: Project Reports

Process Completed By Specification, Development and Certification teams.

Previous Processes Increment Development (E16) or Increment Certification (E17)

Pre-condition None

Subsequent Process Increment Development (E16) or Increment Certification (E17)

Stimuli E4 Complete (I3)

Responses E21 Complete (I14), Project Reports (R13)

State Data Usage The goal of this process is to increase knowledge of the system. For that reason, the engineering files will all be modified during this process, as will Unresolved Questions or Issues, as more questions or issues are resolved. The other state data is used as reference material in order to modify the engineering files. The project schedule in the Project Management Files is used to determine an effort and resource allocation for the process.

Process Description Teams will be prompted to increase their understanding of the problem and solution domain by the appearance of unresolved questions in the state. Performing sufficient iterations of investigation, analysis and documentation to achieve the Completion Conditions by completing the following engineering tasks. These tasks may be conducted concurrently.

1. Read all available literature that may clarify problem and solution domain issues.
2. Solutions found internally within the engineering team will appear in the engineering team files (T1-T5).
3. Interaction with any individuals, including those from outside organizations, that will help increase the understanding of the problem and the solution domain may also be necessary.
4. The solutions/resolutions to these issues will be received as Documents/Working Papers or Informal Communication.
5. Information may be received that is solicited or unsolicited. It is up to the Specification, Development and Certification teams to examine information received in order to determine whether material is useful for their purposes.
6. Any working papers, internal reports, etc. must be placed in the correct Project Document Files. Project Reports are to be submitted externally for publication, as are Working Papers, which are converted into Project Reports.
7. If any internal questions or issues are resolved, the resolutions must be recorded in state data (T6), as well as in any other place that resolution may be preserved.

Management Data Generated Effort, State Data Generated

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process:

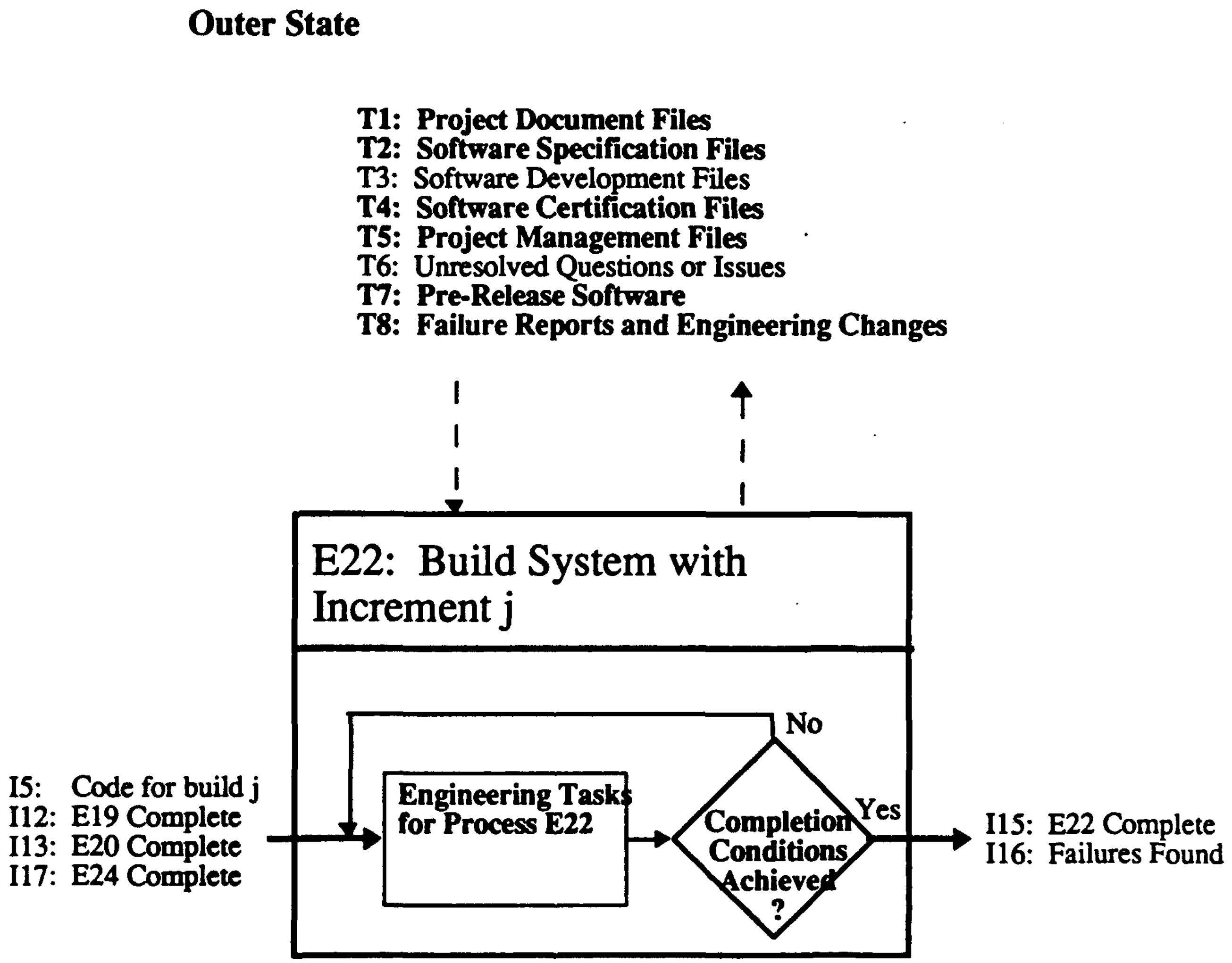
1 Has the project manager determined that a sufficient understanding of the problem and solution domains have been gained to continue with the project?
2. Have solutions that effect the entire team been disseminated?
3. Have all pertinent reviews for this process been completed?
4. Have all action items generated during reviews that pertain to this process been completed?
5. Have all information to be preserved been placed in the correct state data?
6. Have all external responses been submitted?

Keyword References
Review - Section 4.8

Section 4.22 - Process E22 - Build System with Increment j

Process Summary The Build System with Increment j process begins the certification process for the Certification team. Tasks include logging in components, compiling components, assembling the system, and making reliability assessments. The process is illustrated in Figure 4.22.1, and is also described in greater detail below.

Figure 4.22.1 Build System with Increment j Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E22: Build System with Increment j
I5: Code for build j
I12: E19 Complete
I13: E20 Complete
I17: E24 Complete
Engineering Tasks for Process E22
Completion Conditions Achieved ?
No
Yes
I15: E22 Complete
I16: Failures Found

Process Completed By Certification team

Previous Processes Increment Certification (E17)

Pre-condition None for first iteration, Certification Complete? (C14) = FALSE on subsequent iterations

Subsequent Process Increment Certification (E17)

Stimuli Code for build j (I5), E19 Complete (I12), E20 Complete (I13), E24 Complete (I17)

Responses E22 Complete (I15), Failures Found (I16)

State Data Usage Code is put under configuration control during this process, which means Pre-Release Software and Failure Reports and Engineering Changes are modified. The other state data that is used during this process is used as reference material. The project schedule in the Project Management Files is used to determine an effort and resource allocation for the process.

Process Description Complete (achieve Completion Conditions for) the following sequence of engineering tasks during the Build System with Increment j process:

1. Re-assess reliability of the previous version of the system, by summing execution times and failures, accounting for all system versions, taking averages, and predicting the MTTF for the next version, using a Cleanroom Reliability Manager Tool. In effect, complete the reliability analysis of the previous version of the system, in order to prepare for the new version of the system.
2. Log all components that are a part of increment j into the controlled code library. Log all Failure Reports and Engineering Change Notices from the Development team into the Failure Report and Engineering Change File. Keep an accurate count of all new or changed components, in order to calculate engineering changes (a part of reliability assessment). This is to ensure that the state of the system at any time is precisely described. Make sure that every modified component has a corresponding approved Engineering Change Notice. If the cause of the modification was a failure, the Failure Report must also be completed. Modified code that does not have a completed Engineering Change Notice should not be logged in.
3. Compile all new or modified components. If any components do not compile, complete a Failure Report form, and return components, the test scenario and the Failure Report form to the Development team for correction.
4. Build the system by assembling the existing system (increments 1...j-1) and the new components (increment j). If there are any failures while building the system, complete a Failure Report form, and return components, the test scenario and the Failure Report form to the Development team for correction.

Measurement Data Generated Effort, Library Management, Reliability

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process:

1. Have all components necessary to build increment j been received?
2. Are all components under configuration control?
3. Have all components received been compiled successfully?
4. Have all components received been assembled successfully with the existing system into an executable system?
5. Have all faults found during compilation and linking been noted on failure report forms and returned to the Development team?
6. Does every changed component have a corresponding Engineering Change Notice?

7. Is the process complete in regards to what is described in the Construction Plan (Volume 6 of the Specifications)?
8. Have all information to be preserved been placed in the correct state data?
9. Has necessary metrics been gathered?
10. Have all pertinent reviews for this process been completed (for example, if the project is following 2167A, the Function and Physical Configuration Audit needs to be completed)?
11. Have all action items generated during reviews that pertain to this process been completed?

Keyword References

Review - Section 4.8

Reliability, MTTF, Cleanroom Reliability Manager

- "Certifying the Reliability of Software," IEEE Transactions on Software Engineering, January 1986 (Currit, Dyer, Mills)

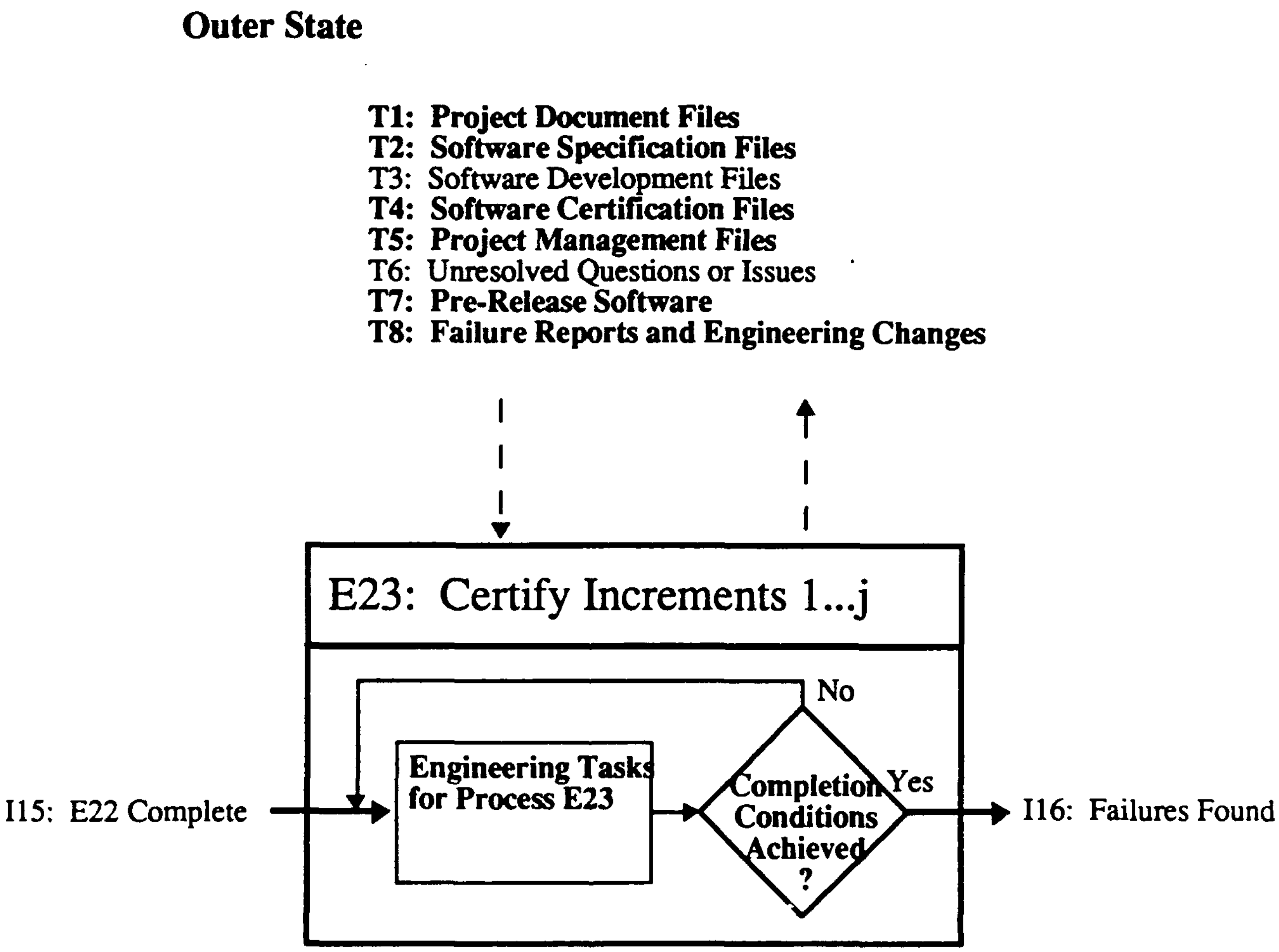
- "Engineering Software Under Statistical Quality Control," IEEE Software, November 1990 (Cobb, Mills)

- "Cleanroom Reliability Manager: A Case Study Using Cleanroom with Box Structures ADL," CDRL 1880, IBM P.O. 308334-PJ, May 1990 (Poore, Mills, Hopkins, Whittaker)

Section 4.23 - Process E23 - Certify Increments 1...j

Process Summary The Certify Increments 1...j process is where the Certification team conducts the Certification of the accumulated increments of code. This process includes executing test scenarios and validating them. The process is illustrated in Figure 4.23.1, and is also described in greater detail below.

Figure 4.23.1 Certify Increments 1...j Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E23: Certify Increments 1...j
I15: E22 Complete
Engineering Tasks for Process E23
Completion Conditions Achieved ?
No
Yes
I16: Failures Found

Process Completed By Certification team

Previous Process Increment Certification (E17)

Pre-condition Pre-Certification Failure (C11) = FALSE

Subsequent Process Increment Certification (E17)

Stimuli E22 Complete (I15)

Responses Failures Found (I16)

State Data Usage The Pre-Release Software is certified during this process, which means Pre-Release Software is used and Failure Reports and Engineering Changes is modified. The Software Certification Files are also modified to as test results appear. The other state data that is used during this process is used as reference material. The project schedule in the Project Management Files is used to determine an effort and resource allocation for the process.

Process Description Complete (achieve Completion Conditions for) the following sequence of engineering tasks during the Certify Increments 1...j process:

1. Select a batch (1 or more) test scenarios at random.
2. Execute tests in the batch in a random order on the current version of the software. Record the time of the test or the time to failure, as the case may be.
3. Validate each test scenario, comparing the actual result to the expected result for that test scenario.
4. If the results do not agree, fill out a Failure Report.
5. If the results do agree, the test scenario passed, which means another test execution is then validated, until the entire batch of test scenarios have been executed and validated.
6. Failures Reports, test scenarios and code are submitted together to the development team for the resolution of any failure.
7. The Certification team may stop certification when a number of failures have been observed, all test cases are executed, reliability goals have been met or reliability goals have not been achieved.
8. If certification is to continue and failures have been found, then failure reports are submitted to the Development team.
9. The appropriate MTTF measure for each execution must be noted in order to allow for the calculation of system reliability.
10. Assess reliability of the present version of the system, if desired, by summing execution times and failures, accounting for all system versions, taking averages, and predicting the MTTF for the next version, using a Cleanroom Reliability Manager Tool.

Measurement Data Generated Effort, Reliability

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

1. Have all test executions been validated with the expected result for that test case?
2. Has testing been stopped for one of the following reasons:
   a) All test scenarios executed,
   b) Observed failures to be corrected by Development team,
   c) Management decision?
3. Have all failures found by the certification team led to a Failure Report being put in the Failure Report File, which will lead to Development team resolution of the Failure Report?
4. Is the process complete in regards to what is described in the Construction Plan (Volume 6 of the Specifications)?
5. Have all additions, deletions or modifications to the state data been completed?

6. Have all necessary measures been gathered?
7. Have all pertinent reviews for this process been completed?
8. Have all action items generated during reviews that pertain to this process been completed?
9. Have all information to be preserved been placed in the correct state data?

Keyword References

Review - Section 4.8

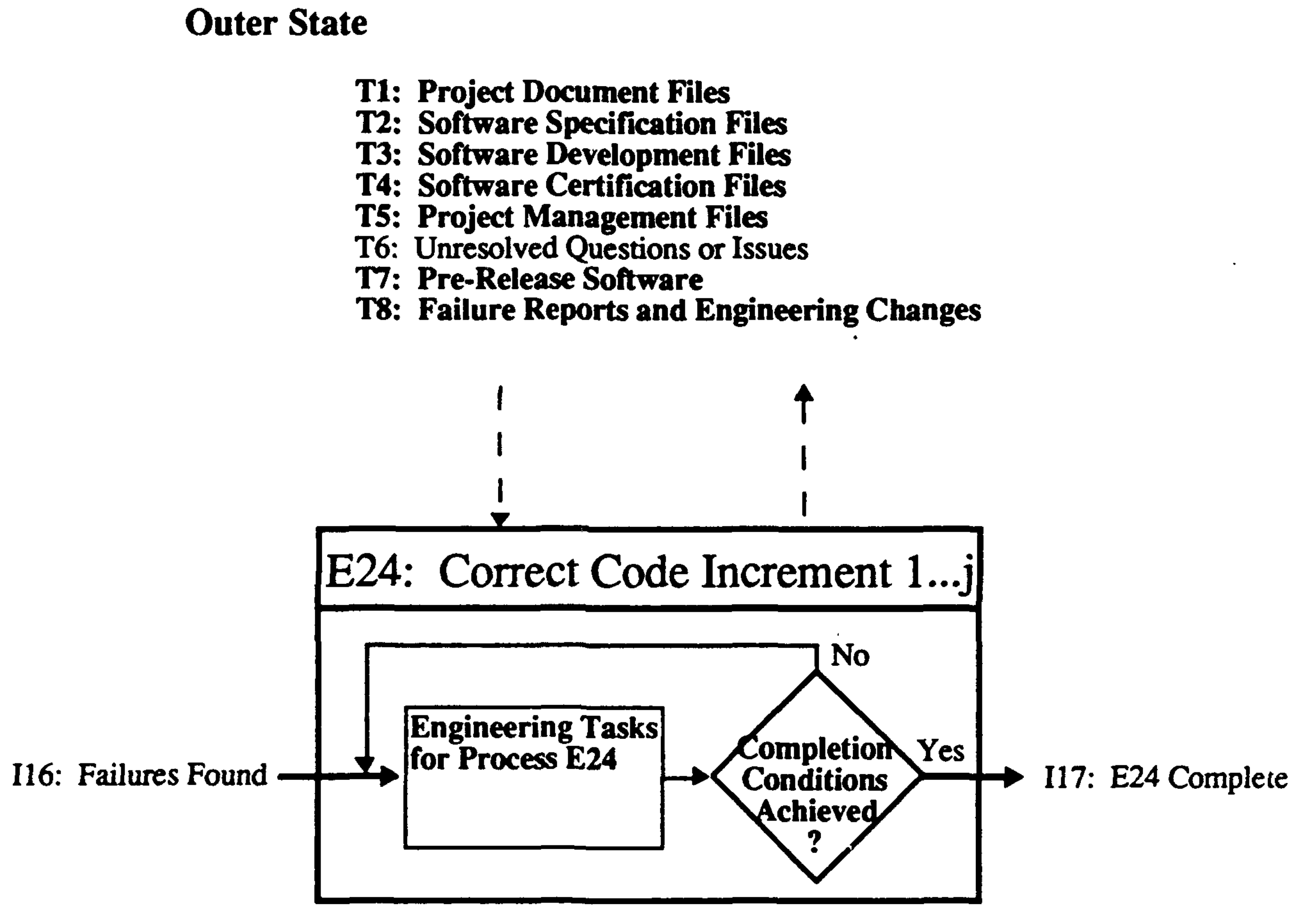
Certification, Reliability, MTTF, Cleanroom Reliability Manager

- Section 10
- "Certifying the Reliability of Software," IEEE Transactions on Software Engineering, January 1986 (Currit, Dyer, Mills)
- "Engineering Software Under Statistical Quality Control," IEEE Software, November 1990 (Cobb, Mills)
- "STARS - Cleanroom Reliability: Cleanroom Ideas in the STARS Environment," CDRL 001 for Task IR-70, September 1990 (Poore, Mutchler, Mills)
- "Cleanroom Reliability Manager: A Case Study Using Cleanroom with Box Structures ADL," CDRL 1880 for Task IR-70, May 1990 (Poore, Mills, Hopkins, Whittaker)

Section 4.24 - Activity E24 - Correct Code Increment 1...j

Process Summary The Correct Code process resolves failures that the Certification team has found in the code. The process is illustrated in Figure 4.24.1, and is also described in greater detail below.

Figure 4.24.1 Correct Code Increment 1...j Process


Outer State
T1: Project Document Files
T2: Software Specification Files
T3: Software Development Files
T4: Software Certification Files
T5: Project Management Files
T6: Unresolved Questions or Issues
T7: Pre-Release Software
T8: Failure Reports and Engineering Changes
E24: Correct Code Increment 1...j
No
Engineering Tasks for Process E24
Completion Conditions Achieved ?
Yes
I16: Failures Found
I17: E24 Complete

Process Completed By Development team

Previous Processes Increment Certification (E17)

Pre-condition Pre-Certification Failure (C11) = TRUE or At Least One Failure (C12) = TRUE and Continue With Certification (C13) = TRUE

Subsequent Process Build System With Increment j (E22)

Stimuli Failures Found (I16)

Responses E24 Complete (I17)

State Data Usage Code is modified during this process, but the controlled version is not modified, which limits change to the Software Development Files. All other state data that is used during this process is used in a reference capacity. The project schedule in the Project Management Files is used to determine an effort and resource allocation for this process.

Process Description Correct failures found by the Certification team, using the following sequence of engineering tasks. The process is completed when the Completion Conditions are achieved.

1. Get a copy of the code, test scenario and Failure Report from the Certification team.
2. Isolate the failure using the Failure Report, test scenario and code. If it is determined that a failure did not actually occur, it must be justified.
3. Make all modifications associated with the resolution of the failure.
4. Verify the correction made, making any additional changes necessary.
5. Determine the number and severity of engineering changes made when resolving any failure.
6. Complete an Engineering Change Notice for each engineering change made to the code.
7. Integrate the change made in the code into all levels of design.
8. Return the modified code, a completed copy of the Failure Report indicating the resolution of the failure and a completed copy of an Engineering Change Notice signed by the project manager, to the Certification team.
9. If changes need to be made to the code that are not prompted by a Failure Report, the development team must justify the changes, before they have been made and verified, to the Change Control board. Only after the change has been approved, can the change be made and given to the Certification team, along with a completed and signed Engineering Change Notice. All levels of design must also be modified to reflect the change.
10. Remember, the code being changed is not the Pre-Release Software. It is actually a copy of code in that library, since the Certification team is the only team to modify the code. They will integrate code submitted by the developers during the Build System with Increment j activity, which will lead to the state data being modified.

Measurement Data Generated Effort

Completion Conditions Each of the following questions must be answered affirmatively in order to complete this process.

1. Have all Software Failure Reports received by the Development team been resolved?
2. Have engineering changes made to the code been carefully noted?
3. Have all changes to the code been verified by Development team members, as well as by the Development team as a whole?
4. Has every change made to the code also been propagated back throughout the clear, state and black box designs?
5. Has an Engineering Change Notice been completed for every modification to the code?
6. Have all pertinent reviews for this process been completed?
7. Have all action items generated during reviews that pertain to this process been completed?
8. Have all information to be preserved been placed in the correct state data?

Keyword Reference

Review - Section 4.8
Sections 9.4, 9.5 and 9.6

Section 5
State Data For Cleanroom Engineering

5.0 State Data Summarized

Engineers performing each of the Cleanroom Engineering processes during a software development project work primarily with the files that contain the state data which define the current state of the project. At each stage, the engineers find the information they require in the state data files. Then, they use their intellect guided by the Cleanroom Engineering processes to make a next invention and record the invention along with its rationale, and where applicable, a verification to form the new state. This process continues until the design is complete. The engineers make small inventions followed by verifications where possible, until the completed software and other deliverables are available. When the software is complete and certified, the state is used to produce the final responses in terms of distribution software, maintenance software, final specification, user documentation, archived documentation and management metrics. During the course of the project, the project team needs to produce intermediate results used by people outside of the project to assess progress and adequacy of the design to determine if the software will meet its design objectives. The intermediate results need to be extracted from the state data files where they reside.

The state data then plays a very essential role in a software project. It is vital that the project team and all other interested parties agree on a format for the files containing the state data and that the project team be meticulous in maintaining all project information in the state data files in the prescribed form. In this way everyone associated with the project can proceed with confidence.

A good analogy is the financial records of a company. The reason accountants can produce financial accounts that everyone generally accepts as representing the current state of the financial progress of the company is that they meticulously maintain files containing the state data reflecting all the transactions. The accountants then follow processes to extract the required information for some report. The process works because it is orderly and followed meticulously. The same can be true for software projects. The significant difference between accounting and software development is that in accounting, the transactions that are tracked in the state files come from outside the accounting department. In a software project, however, the transactions reflected in the state files result from inventions made be the software engineers. But the need for meticulous record keeping is the same.

Control must be kept over the state data. Specifically, state data should be deleted when it is superseded, or it can easily become an overwhelming body of information. The policies as to when and how material is removed from state data is an organizational decision that must be thought out carefully. Change to state data needs to be done in a controlled manner. These issues both need to be considered as a part of organizational policy, dependent on the procedures and constraints found in each organization. Both issues are critical but are not addressed here due to the organization-specific nature of these issues.

Software project state data can be maintained in eight files as summarized in this section. The following eight sections state requirements for the maintenance and storage of each class of state data so individual organizations can then use this information to organize storage and maintenance procedures for state data.

T1: Project Document Files

The Project Document Files contain a library of documents that are relevant to the project. The library contains six types of documents. The types are:

Project Document Index — The project document index, which organizes the information found in the Project Document Files to allow multiple views of documents. This may be as simple as a list next to a bookshelf, or as sophisticated as an on-line system with thesaurus and multiple modes of access to material.

External References — These are documents that are prepared by groups outside the project. They have either been delivered to the project or have been found by the project staff. In either case, these documents have been stored in the document library.

Project Reports — These documents are reports that have been prepared by a member of the project team and have been published outside the project.

Working Papers — These documents have been prepared by a member of the project team and are internal to the project team. Working papers consist of any document archived by the project team, usually by associating a document number with the item. Only material that has a document number with it will be kept in state.

Trade Studies — These documents are specific documents that evaluate the relative value between alternatives using a quantitative comparison scheme. Trade studies included in the Project Document files are prepared by the Specification team.

Project Review Minutes — These documents define conclusions and action items that have been allocated at referenced project review meetings.

T2: Software Specification Files

The Software Specification Files contain the official copy of the specifications for the software. These files are maintained in electronic form using a desk top publishing system. The specifications are in six volumes as follows:

1. The Mission Volume which defines the mission the software is to perform.

2. The Functional (Black Box) Specification Volume which defines an implementation-free internal view of the software in terms of Black Box functions. The Black Box functions define all responses in terms of stimuli histories.

3. The Functional Specification Verification Volume which justifies the correctness of the Functional (Black Box) Specification.

4. The Users Reference Manual Volume which defines all the external inventions that define the software from its user's perspective.

5. The Usage Profile Volume which defines the expected usage profile of the software by its users in the planned usage domain.

6. The Construction Plan Volume which defines the specific modules that will be in each build that the Development team will develop and the Certification team will certify in order to complete the project.

Additionally, an accounting of all specification changes and change requests needs to be maintained. This can be done as a part of the change control of the specifications or separately.

T3: Software Development Files

The Software Development Files contain eight classes of objects as follows:

Design Index
The Design Index defines all references to design objects. Design objects include state data, external stimuli, external responses, boxes, internal stimuli, internal responses.

Trade Studies
Trade studies evaluate the relative value between alternatives using a quantitative comparison scheme. Trade studies included in the Software Development files are prepared by the Development team.

Boxes
Boxes define the Black, State and Clear Box transition formulas for each box in the usage hierarchy.

Verifications
Verifications record the verification arguments for box expansions and code expansions.

Code Refinement
Code refinements define the expansions in terms of code as Clear Box functions are expanded into code.

Design Notes
Design notes are used to record any idea or concept that the engineers feel should be recorded as the design progresses.

Code
Code is the final product of the Development team. When the Development team is complete with an increment and satisfied that the code meets the specifications, it turns the code over to the Certification team for certification.

Metrics
The running count of boxes developed must be collected here. This may be done separately or as a part of the Boxes section above.

T4: Software Certification Files

The Software Certification Files contain eight classes of objects as follows:

Certification Index

All material found in the Software Certification Files is cross-referenced in order to allow easy access. Test scripts, sections of the sampling plan, test scenarios, solutions and results may need to be viewed in a variety of ways.

Certification Notes

Any material that pertains to the certification of the software, but does not fit into the other categories below, will appear here. Specifically, notes, investigations, and thoughts pertaining to the certification process are considered to be certification notes.

Sampling Plan

The test plan defines the sampling plan that has been developed to estimate the reliability of the software to be certified.

Script Generators/Test Scripts

Test scripts and test script generators are descriptions of state transitions in terms of stimuli distributions. The scripts and script generators allow the Certification team to generate test scenarios that have been randomly generated for usage testing. These test scripts fragments are used to prepare complete test scenarios.

Test Scenarios

Test scenarios are scripts that guide a tester in running a certification test. They define the next input and expected output.

Solutions

Test solutions are recorded as part of the test scenarios.

Results

Results define and record the results that were produced by the software for a named test scenario of each version of the software for which that test case was selected to be run. The final result is a determination of pass/fail and the measured MTTF for the test. MTTF is recorded in units meaningful for the software being certified.

Metrics

A number of measures must be collected here. The measures include: Test scenarios developed, Numbers of components, Sizes of components, Numbers of new components, Numbers of modified components, Number of compiled components, Compilation CPU time, Number of assembled components, system dates for each version, Execution time per test scenario, Failure report numbers per test scenario, System version for each test scenario, and system reliability. The data may be collected separately or as parts of the sections above.

T5: Project Management Files

The Project Management Files contain the following six types of documents:

Project Charter

The Project Charter defines the project. It is a stimulus from management, and must be agreed to by both management and the Engineering team.

Master Project Schedule

The Master Project Schedule is the schedule given by management to the Engineering team, which outlines the deadlines and milestones that the Engineering team must achieve in order to complete the project.

Engineering Activity Records

The Engineering Activity Records include all information that the Engineering team submits as status reports. This information allows the Engineering team to manage its own effort, as well as generating information that will allow management to assess Engineering team progress.

Project Schedule

The Project Schedule handles the allocation of all resources (human, computer, etc.) according to the schedule organized to produce all items during the life cycle. Of course, the content of the schedule will change over time. The schedule will give specific personnel explicit assignments for a scheduled period of time. In this manner, all issues pertaining to time with the project are archived here. It takes, as a start, the Master Project Schedule and adds Engineering team specific information.

Task Assignments

All task assignments made to all Engineering team members must be kept. The list of task assignments will allow any staff member to know exactly what tasks that person needs to perform.

Metrics

The Hours per process, tasks assigned, tasks completed, schedule modifications and productivity measures are collected and stored here. This may be done separately or as a part of the other sections found above.

T6: Unresolved Questions or Issues

Each time the project team submits a question to gain more information, test a hypothesis, etc., the question or issue is recorded in the Unresolved Question or Issues file. When the resolution is received, it is recorded in the same file or is given a reference that can be found in the Project Document Files.

T7: Pre-Release Software

The Pre-Release Software file is the library of software that has been released to the Certification team by the Development team for certification. This file is maintained by the Certification team using rigorous configuration management techniques. After the Development team has turned over an increment to the Certification team, if the Development team requires a copy of some of the software to remedy an observed failure, they request a copy from the Certification team. The fix for the observed failure is returned to the Certification team by the Development team as an engineering change which specifies the changes required to each affected software module. The Certification team applies the changes and resumes the certification process with the new version of the software. The Certification team maintains a library of the source code and an executable version of the software that they use for testing. Pre-Release software files are electronic files that can be maintained using any applicable software library system.

T8: Failure Reports and Engineering Changes

Each failure encountered by the certification as they measure the reliability of the software is recorded in a failure report. The Software Failure Report contains the following information:

Test Scenario Identification

Description of Test Scenario

Type of Failure

Location of Failure in Scenario

Test Results Prior To Failure

Description of Observed Failure

Specification Reference For Observed Failure

Modules suspected causing failure (from information provided with the test output, if any)

Attachments as required

When the failure is resolved, the identification of the engineering change that is intended to fix the failure is noted on an Engineering Change Notice. The Engineering Change Notice contains the following information:

Identification of Failure Report(s) Fixed By This Engineering Change Notice

Textual Description of Change

Routines Modified By Engineering Change Notice

Definition Of Changes Made To Each Routine

Source of Error (Specifications, Black Box, State Box, Clear Box, Code, Previous Modification)

Statement that all required files have been adjusted to reflect change

State data is updated and used in various Cleanroom Engineering processes. Table 5.0.1 defines state data usage by the Cleanroom process.

In the following sections, requirements are provided for the maintenance of each class of state data along with advice on how the state data is used or prepared. The intent is that individual organizations can then use this information to organize the best mode of state data storage by matching their resources and the needs of the project.

5.1 Project Document File

5.1.1 Organization and Procedures

The Project Document File receives reference materials from a variety of sources. The format, content and importance may vary a great deal from document to document. The primary needs are to keep the organization and integrity of the information and keep it easy to reference and use. For these reasons, the Project Document File needs to be considered a library, and treated as such.

First, an organizational scheme must be determined. Each document must be separately accessible, in a feasible manner. For that reason, an index is necessary, which lists documents that pertain to, or are related to, a certain topic. This could simply be a keyword list, or can be a more sophisticated thesaurus. Additionally, the five types of documents must also be kept separate, maybe with a unique indexing scheme, so the purpose of a specific document is not lost.

To keep the integrity of the information, all unique versions of a document must be recorded. For example, draft and final versions of a trade study must be kept distinct, to prevent the use of incorrect reference information. Typically, it is useful to provide a periodic inventory of the Project Document File, in order to ensure that the correct version of any document is accessible. Access to the file must be controlled. Of course, every member of the Engineering team needs to have access to the entire Project Document File. Therefore, when materials are removed from the file, temporarily or permanently, they must be signed out, to keep màterial accessible and enforce accountability for resources. In this way a staff member can approach

Table 5.0.1 State Data Usage By Process

	T1	T2	T3	T4	T5	T6	T7	T8
E0: Cleanroom	M				M	M		
E1: Project Invocation								
E2: Program Management								
E3: Project Information Management	U	U	U	U	U	U	U	U
E4: Software Solution Specification								
E5: Software Development and Certification	U	U	U	U	U	U	U	U
E6: Prepare Final Project Releases	U	U	U	U	M	U	U	U
E7: Maintain Project Schedule	U	U	U	U	U	U	U	U
E8: Prepare For and Conduct Project Review	U	M	M	M	M	M	U	U
E9: Prepare and Submit Project Status Reports	M				M	M		
E10: Receive Stimuli	U	U	U	U	U	M		
E11: Submit a Question or Issue	M	U	M	M	U			
E12: Understand Problem Domain	M	U	M	M	U			
E13: Understand Solution Domain	U	M			U			
E14: Write Specifications	U	M			U			
E15: Write Construction Plan								
E16: Increment Development								
E17: Increment Certification	U	U	M		U			
E18: Develop Increment i	U	U		M	U			
E19: Develop Certification Tests for Increments 1..i								
E20: Update Specifications	U	M	U	U	U	U	U	U
E21: Increase Understanding of Problem and Solution Domains as Required	M	U	M	M	U	U	U	U
E22: Build System with Increment j	U	U		M	U		M	M
E23: Certify Increments 1..j								
E24: Correct Code	U	U		M	U		U	M
Increments 1..j	U	U	M	U	U		U	U

Key:
M means that the state data is modified by the process
U means that the state data is used by the process but not modified

a co-worker to look at a document which the co-worker has borrowed from the file. It is often desirable to place time limits on the borrowing of materials to assist in the inventory process.

There are a number of ways by which the file can be made accessible. First, material may be kept on a variety of media. External references may be on paper, while trade studies may be on-line, with project review minutes appearing on videocassette. The easiest medium for any staff member to use is the most valuable. Second, an Engineering team member must be made aware of material as it is placed in the file. A weekly status report of all new material in the library, or a requirement for Engineering team members to look at the updated index are possibilities. Finally, the material in the file must be placed in a location accessible to the Engineering team members. A file, albeit very sophisticated, is useless if it is kept in a far-away location. The file is only useful if it is used. For this reason, on-line access may be optimal for large projects while less expensive means, like notebooks, serve the needs of medium-sized projects.

A number of schemes and rules were given which suggest a manner in which the Project Document File can be used. The specific manner in which it is used is dependent on the organization. Some organizations may be able to increase use of on-line storage, while others may find an off-line scheme easier. As long as the primary issues of organization, integrity and ease of use are enforced, the Project Document File can be well used.

5.1.2 Content

There are a total of six items in the Project Document File. They are the Project Document Index, External References, Project Reports, Working Papers, Trade Studies and Project Review Minutes.

Project Document Index

The project document index, organizes the information found in the Project Document Files to allow multiple views of documents. This may be as simple as a list next to a bookshelf, or as sophisticated as an on-line system with thesaurus and multiple modes of access to material.

External References

These are documents that are prepared by groups outside the project. They have either been delivered to the project or have been found by the project staff. In either case, these documents have been stored in the document library.

Project Reports

These documents are reports that have been prepared by some member of the project team and published outside of the project.

Working Papers

These documents have been prepared by a member of the project team and are internal to the project team. Working papers consist of any document archived by the project team, usually by associating a document number with the item. Only material that has a document number with it will be kept in state. For example, the software development plan will appear as a working paper in this file.

Trade Studies

These documents are specific documents that evaluate the relative value between alternatives using a quantitative comparison scheme. Trade studies included in the Project Document files are prepared by the Specification team.

Project Review Minutes These documents define conclusions and action items that have been allocated at referenced project review meeting.

The use of and modification to each section of the Project Document File by each process is listed in Table 5.1.1.

5.2 Software Specification Files

5.2.1 Organization and Procedures

When designing software to fulfill a mission, there are two major phases of the project as follows:

The **Specification Phase** where six interrelated decisions and/or inventions are made and recorded in terms of a specification for the software. The six decisions/inventions are:

defining and agreeing on the mission the software is to fulfill (Very often the mission statement is thought of as requirements which specify what the software shall do. In general, the mission statement for software is derived from a larger requirement or mission in that software very rarely is the only component of the system — there are typically people and/or devices with whom the software works to fulfill the mission of the entire system.),

inventing the external behavior of the software in terms of (1) stimuli to the software, (2) the responses from the software, and (3) the control structure that defines when each response should be produced in terms of stimuli histories that will enable the software to fulfill its assigned mission(s),

justifying how that external behavior of the software will fulfill the desired mission of the software,

estimating how the software will be used to fulfill its assigned mission so a usage profile can be established which is required to estimate how reliable the software as constructed can fulfill its assigned mission, and

specifying the hardware/software environment in which the software will be operating.

planning the schedule for deliverables and milestones which the Engineering team will strive to fulfill.

The **Development and Certification Phase** is where the software is developed in accordance with the specification and then the software, as constructed, is measured to determine how well it conforms to the specification when the software is used as planned.

The specification document is the document that records all the inventions and decisions made during the specification phase and also records the arguments why the software will fulfill its intended function if it is built in accordance with the specifications. The specification document typically goes through many iterations as the external inventions are made and refined. It is important that each version of the specifications be formal, well written and precise, even when the specifications are preliminary. The team preparing the specifications should regard their task as the development of a document that is to be published.

Table 5.1.1 Project Document State Data Usage by Process

	Index	External References	Project Reports	Working Papers	Trade Studies	Project Review Minutes
E0: Cleanroom						
E1: Project Invocation	M			M		
E2: Program Management						
E3: Project Information Management						
E4: Software Solution Specification	U	U	U	U	U	U
E5: Software Development and Certification						
Prepare Final Project	U	U	U	U	U	U
E6: Releases						
E7: Maintain Project Schedule	U	U	U	U	U	U
E8: Prepare For and Conduct Project Review	U	U	U	U	U	U
E9: Prepare and Submit Project Status Reports	U	U	U	U	U	U
E10: Receive Stimuli	M	M				M
E11: Submit a Question or Issue	U	U	U	U	U	U
E12: Understand Problem Domain	M	U	M	M		
E13: Understand Solution Domain	M	U	M	M	U	
E14: Write Specifications	U	U	U	U	U	U
E15: Write Construction Plan	U	U	U	U	U	U
E16: Increment Development						
E17: Increment Certification						
E18: Develop Increment i	U	U	U	U	U	U
E19: Develop Certificrtion Tests for Increments 1..i	U	U	U	U	U	U
E20: Update Specifications	U	U	U	U	U	U
E21: Increase Understanding of Problem and Solution Domains as Required	M	U	M	M	M	U
E22: Build System with Increment j	U	U	U	U	U	M
E23: Certify Increments 1..j	U	U	U	U	U	U
E24: Correct Code Increments 1..j	U	U	M	U	U	U

Key:
M means that the state data is modified by the process
U means that the state data is used by the process but not modified

In many software projects specifications are not given the attention they require. There is a great hurry to start building the software because so many problems are anticipated with the development. This has the very unpleasant consequence of introducing change which causes rework, resulting in users often settling for a product they are not really happy with. One reason for starting with the software development early (that is, before the specifications are available) is because when the heuristic software development methods associated with contemporary software engineering are used, it has been observed that very often the result will not be the desired result. It is then necessary to rework the software. With this anticipation, it is not unusual to start building software before the specifications are complete. A self-fulfilling prophecy has been created. When the rigorous construction methods associated with Cleanroom Engineering are used, these difficulties are not likely to occur. Another reason software construction is started early is to accommodate bottom-up exploration to determine the look and feel of the software. In many cases this bottom up exploration is required. It should be regarded as just that - engineering modeling. When engineering modeling is separated from top-down design, the result is high-quality bottom-up exploration and a high quality final product.

5.2.2 Content

There are many possible frameworks in which the specifications can be recorded. It is important that the project team agree on a framework for the specifications and develop a formal, precise document in accordance with the agreed-upon framework. To help project teams specify a framework for the specification document, a six volume framework is proposed. A sample outline for the six volumes is given in Figure 8.2.1. Each of the six documents has a specific set of purposes, which define the content and format of the volume.

The purposes of the Mission Volume are (1) to define the mission that the software is to fulfill, (2) to define the context in which the software will be operating and (3) to record the argument that says if the software satisfies the defined mission, it will accomplish its part of the goals of the automation.

This User's Reference Manual volume contains a precise definition of all the inventions made by the software designers in terms of the stimuli and responses that have been invented for the software to interface with the environment in which it will be operating. A user with sufficient subject matter expertise should be able to use the software by using this volume.

The purpose of the Functional (Black Box) Specification Volume is to define an implementation-free view of the internal structure of the software that is being designed. This implementation-free view is obtained by specifying functions which define the conditions in terms of stimuli histories that cause the presentation of each possible response which can be produced by the software in terms of stimuli histories.

The Functional Specification Verification Volume presents an argument which justifies the correctness of the Functional (Black Box) Specification Volume. This argument is not easy to make, but serves as the basis for accepting or rejecting the specifications. This view switches the responsibility for showing that the specifications will solve the problem to the specification developer from the user, which is the more traditional approach.

The Expected Usage Profile volume contains the definition of how it is anticipated the software will be used by each class of user. This definition is required to develop test scenarios in accordance with usage specifications so it is possible to make measurements for how reliable the software will be when it is put into service in the environment where it is expected to operate.

The purpose of the Construction Plan volume is to develop a plan for the Development and Certification teams, during actual product development. The plan contains the actual modules and functions that will be a part of each increment. This will present the list of modules that the Development team needs to produce for any increment, and will outline the functionality available for the Certification team to certify for an increment.

The use of and modification to each section of the Software Specification Files by each process is listed in Table 5.2.1.

5.2.3 Change Control

Controlling change in the specifications is just as important as change control with code. For this reason, specifications must also be under configuration management, with all changes being approved by the specifications team, with outside management assistance, if necessary. Additionally, the project manager should be required to sign any approved request to change the specifications. If necessary, the specifications may need to be maintained in a secure manner (on-line with passwords, or in a locked desk) to prevent unauthorized changes, but these measures should be decided on a case-by-case basis.

5.3 Software Development Files

5.3.1 Organization and Procedures

The Software Development Files serve as the repository for all material used in the design and implementation of the software. Each type of material has a desired structure which, if followed, will allow the material to be more easily used.

With Cleanroom, software is developed in a top-down, iterative process with verification between each inventive step. To facilitate this iterative process, it is important that the material be stored so the relationship between all design objects can be easily determined. The design index serves this purpose, allowing a user of the file to find state data, external and internal stimuli and responses, boxes and their uses.

Every design object must be clearly and uniquely defined. The process is more involved than just ensuring that no two final boxes have the same name. Multiple versions of the same box, which may appear when a box goes through a number of reviews, must be marked with a version number. The same notation must be done for code refinements, code and verifications. If boxes, verifications and code are kept on line, the version numbers may be handled by configuration management systems. It is of utmost importance that all Development team members look at the same version of a design object.

As design objects are accepted, by reviews or verification, old versions may be deleted or removed. This minimizes the extra complexity of searching through dated and obsolete materials. Which versions are deleted is up to the Engineering team manager, and will probably be set as policy early in the project.

Code has a special requirement. Once it is submitted to the Certification team, it must be deleted from the Software Construction File (although an off-line copy may be kept). This prevents accidental

Table 5.2.1 Software Specification State Data Usage by Process

	Specifications	Metrics
E0: Cleanroom		
E1: Project Invocation		
E2: Program Management		
E3: Project Information Management		
E4: Software Solution Specification	U	U
E5: Software Development and Certification		
Prepare Final Project	U	U
E6: Releases		
E7: Maintain Project Schedule	U	U
E8: Prepare For and Conduct Project Review	U	
E9: Prepare and Submit Project Status Reports	U	M
E10: Receive Stimuli		
E11: Submit a Question or Issue	U	
E12: Understand Problem Domain	U	
E13: Understand Solution Domain	U	
E14: Write Specifications	M	
E15: Write Construction Plan	M	
E16: Increment Development		
E17: Increment Certification		
E18: Develop Increment i	U	
E19: Develop Certification Tests for Increments 1..i	U	
E20: Update Specifications	M	
E21: Increase Understanding of Problem and Solution Domains as Required	U	
E22: Build System with Increment j	U	
E23: Certify Increments 1..j	U	
E24: Correct Code Increments 1..j	U	

Key:
M means that the state data is modified by the process
U means that the state data is used by the process but not modified

modifications to controlled code. For example, if a developer makes a failure correction to a version of code that exists in the non-controlled library, which may have been modified in an experimental manner, that experimental change may go into the controlled library without an associated Engineering Change. As will be stated in the description for Pre-Release Software (T8), Development team members can get copies of the code at any time, but change can be controlled by the Certification team, by not updating routines unless Engineering Changes are attached to them. This minimizes accidental modifications to controlled code.

The material in the Software Development Files should not be viewed by the Certification team. Seeing a design or code for a part of the system may bias the certification process, which will also bias the reliability estimate. Efforts need to be made to prevent these accidental biases.

5.3.2 Content

The Software Development Files are developed by the Development Team using the engineering practices described in Section 9. They contain:

Design Index

The Design Index can be on-line or off-line. An automated means of updating the Design Index minimizes the time spent manually managing the Index. A convenient way to access the entire design trail is critical when using Cleanroom; only by having the complete design trail can one determine what does or does not need to be modified. Typical contents of Design Index for each Design Object appear as follows:

State Data		
	Name	
	Stimuli history represented	
	Definition of abstraction	
	Where computed	box name
		sub-function name
	Where used	box name
		sub-function name
External Stimuli		
	Name	
	Where defined	users reference manual reference
	Where processed	box name
		sub-function name
External Responses		
	Name	
	Where defined	users reference manual reference
	Where used	box name
		sub-function name
Internal Stimuli		
	Name	
	Purpose	
	Where defined	box name
	Where used	box name

Internal Responses
- Name
- Purpose
- Where defined box name
- Where used box name

Boxes
- Name
- Purpose
- Level in usage hierarchy
- Parent Box(es)
- Box(es) used
- State Data used
- Responses used
- Stimuli used

Trade Studies

Trade studies are a critical part of engineering. Systems engineers conduct and document many trade studies. Software developers do not use trade studies as frequently. With Cleanroom practices, software developers now have the opportunity to improve their engineering by conducting trade studies. Software developers will want to use the format for trade studies that have been adopted by their organization.

Boxes

Boxes define the Black, State and Clear Box transition formulas for each box in the usage hierarchy. Figures 9.2.1 through 9.2.3 illustrate the box structure algorithm and the templates for Black, State and Clear Boxes. This section of state data must preserve the entire specification and design trail; i.e., the content of every iteration of the Box Structures must appear here.

Verifications

Verifications record the verification arguments for box expansions and code expansions. Figures 9.5.1 and 9.5.2 present the Correctness Theorem, and the rules for functional verification of the basic structures. Every program can be developed from the basic structures, thus only a limited number of verification constructs are necessary.

Code Refinement

Code refinements define the expansions in terms of code as Clear Box functions are expanded into code. Figures 9.4.1 and 9.4.2 illustrate the concept of code refinement.

Design Notes

Design notes are used to record any idea or concept that the engineers feel should be recorded as the design progresses.

Code

Code is the final product of the Development team. When the Development team is complete with an increment and it is satisfied that the code meets the specifications it turns the code over to the Certification team for certification. Figure 9.4.4 defines a set of rules for documenting code.

Metrics

The total numbers of boxes produced must be tracked in order to assist the process and product assessment and improvement activities. This may also be handled in some automated means in the "Boxes" section of this file.

Rules and templates such as those defined in Section 9 need to be tailored to each organization. Initially, an organization may want to start with these rules and templates, eventually extending them to meet their special needs.

The use of and modification to each section of the Software Development Files by each process is listed in Table 5.3.1.

5.4 Software Certification Files

5.4.1 Organization and Procedures

The Software Certification Files are developed by the Certification Team using the engineering practices described in Section 10. The Software Certification Files serve as the repository for all of the material used in the certification of the software. Each type of material has a desired structure, which if followed, will allow the material to be more easily used.

The Software Certification Files need to be kept organized for a number of reasons. First, the testing approach used is a statistical testing approach. If materials are not kept in an organized manner, data may be lost that will bias the certification estimates. Additionally, the entire approach must be clearly defined and followed to ensure that the test scenarios are generated according to the operational profile of the system. Second, each test scenario, with its corresponding result, must be preserved across test executions to preserve data necessary for reliability estimates, to determine the status of the certification effort, and to keep control of retests. Finally, since the test process is solely the concern of the Certification team, material must be kept in a manner which will keep the security of the material. A developer who unintentionally sees a test scenario, may bias his/her development effort, to account for a specific test scenario. This also biases the reliability estimate, since the test case is no longer random.

5.4.2 Content

Certification Index

The certification index may appear as follows. The intent is for the Certification team to have easy access to all of the certification information.

```
Test Script a
    Used In
        Test Scenario a
        Test Scenario b
Test Script b
    .....
```

Table 5.3.1 Software Development State Data Usage by Process

	Design Index	Trade Studies	Boxes	Verifi-cations	Code Refinement	Design Notes	Code	Metrics
E0: Cleanroom								
E1: Project Invocation								
E2: Program Management								
E3: Project Information Management								
E4: Software Solution Specification	U	U	U	U	U	U	U	U
E5: Software Development and Certification								
E6: Prepare Final Project Releases	U	U	U	U	U	U	U	U
E7: Maintain Project Schedule	U	U	U	U	U	U	U	U
E8: Prepare For and Conduct Project Review	U	U	U	U	U	U	U	
E9: Prepare and Submit Project Status Reports	M		U	U	U	U	U	M
E10: Receive Stimuli								
E11: Submit a Question or Issue	U	U	U	U	U	U	U	U
E12: Understand Problem Domain	M	M				M		
E13: Understand Solution Domain	M	M				M		
E14: Write Specifications								
E15: Write Construction Plan								
E16: Increment Development								
E17: Increment Certification								
E18: Develop Increment i	M	M	M	M	M	M	M	
E19: Develop Certification Tests for Increments 1..i								
E20: Update Specifications	U	U	U	U	U	U	U	U
E21: Increase Understanding of Problem and Solution Domains as Required	M	M	U	U	U	U	U	U
E22: Build System with Increment j								
E23: Certify Increments 1..j								
E24: Correct Code Increments 1..j	M	M	M	M	M	M	M	

Key:
M means that the state data is modified by the process
U means that the state data is used by the process but not modified

```
Script Generator a
    Used In
        Test Scenario a
        Test Scenario b
Script Generator b
    .....

Test Scenario a
    Increment number x
    Uses
        Script Generator a
        Script Generator b
    Contains
        Test Script a
        Test Script b
    Solution
        Solution a
    Result
        Result a
Test Scenario b
    .....
```

Certification Notes

Certification notes include any item generated by the Certification team. The notes include such topics as the results of investigations, thoughts, and ideas pertaining to the certification of the software product.

Sampling Plan

The sampling plan defines how to draw usage tests from the entire population of possible uses in order to certify the software. See Section 10.1 for factors that influence the preparation of the sampling plan.

Script Generators/Test Scripts

Script generators are executable code components which generate test scripts. Generators are necessary when a specific state transition can be completed in a variety of ways, due to variability of data, or different possible stimuli. Figure 10.2.2 shows a sample test script generator.

Test scripts are the actual commands that result in state transitions in a test scenario. These are actual stimuli sequences. Figure 10.2.3 presents a sample test script.

Test Scenarios

A test scenario is developed from a sequence of randomly generated state transitions. The state transitions can be selected using a random number generator. The test scenario is a sequence of test scripts which implement the above-mentioned state transitions. Figure 10.2.4 presents an example for a test scenario.

Solutions

A solution is the expected result of a test scenario. Figure 10.3.1 provides a typical form for recording the solution manually. The exact format of the form will depend on the software being certified. In many instances, it will be worth the effort to develop a means to record the information in an electronic file so it can be easily compared to the results.

Results
Results can take a number of forms. It is critical that the Certification team gather all external responses that are being generated. This may include capturing displays, impulses on a bus, and printer output. If the data can be recorded electronically during execution it may be be easier to appraise solutions.

Metrics
A number of measures must be kept track of in this section. The measures include: Test scenarios developed, Numbers of components, Sizes of components, Numbers of new components, Numbers of modified components, Number of compiled components, Compilation CPU time, Number of assembled components, system dates for each version, Execution time per test scenario, Failure report numbers per test scenario, System version for each test scenario, and system reliability. These measures can be collected in an automated or manual manner as parts of the other sections above in the Software Certification Files, or they may be kept specifically in this section.

The use of and modification to each section of the Software Certification Files by each process is listed in Table 5.4.1.

5.5 Project Management Files

5.5.1 Organization and Procedures

The material in the Management Files is publicly accessible; i.e., any member of the Engineering team may look at the Management Files. The way that much of the material in the Management File will appear is not the choice of the Engineering team. Since much of the material is given by management as a stimulus to the Engineering team, it is not possible for the Engineering team to force a certain structure to be required for a stimulus. Additionally, the Project Status Reports, which are a response from the Engineering team, will probably be in a certain format, that the organization desires.

The project schedule will initially use the Master Project Schedule in order to set management-directed external milestones for reviews, deliverables, etc. Subsequently, the Engineering team will need to set internal deadlines for reviews, internal deliverables, build schedules and activity milestones. The schedule will be modified as the project continues.

In addition to dates, the project schedule also needs to outline the use of personnel for the project. Engineering team members need to be assigned to specific tasks over a specified period of time. Allocation of staff to tasks will also change as the project continues, so the schedule needs to maintained in an easy-to-change format.

The project schedule is needed by all staff members during every activity in order to determine the resources at one's disposal at any moment. The project schedule is modified by the Engineering team manager, as the manager is the person responsible for the project. It may appear on-line or off-line, whichever is more convenient for the organization.

Table 5.4.1 Software Certification State Data Usage by Process

		Cert. Index	Cert. Notes	Sampling Strategy	Test Scripts	Test Scenarios	Solutions	Results	Metrics
E0:	Cleanroom								
E1:	Project Invocation								
E2:	Program Management								
E3:	Project Information Management								
E4:	Software Solution Specification	U	U	U	U	U	U	U	U
E5:	Software Development and Certification Prepare Final Project	U	U	U	U	U	U	U	U
E6:	Releases								
E7:	Maintain Project Schedule	U	U	U	U	U	U	U	U
E8:	Prepare For and Conduct Project Review	U	U	U	U	U	U	U	U
E9:	Prepare and Submit Project Status Reports	M	U	U	U	U	U	U	M
E10:	Receive Stimuli								
E11:	Submit a Question or Issue	U	U	U	U	U	U	U	U
E12:	Understand Problem Domain	M	M						
E13:	Understand Solution Domain	M	M						
E14:	Write Specifications								
E15:	Write Construction Plan								
E16:	Increment Development								
E17:	Increment Certification								
E18:	Develop Increment i								
E19:	Develop Certification Tests for Increments 1..i	M	U	M	M	M	M		
E20:	Update Specifications	U	U	U	U	U	U	U	U
E21:	Increase Understanding of Problem and Solution Domains as Required	M	M	U	U	U	U	U	
E22:	Build System with Increment j	M						M	U
E23:	Certify Increments 1..j	M		U	U	U	U	M	U
E24:	Correct Code Increments 1..j	U				U	U	U	

Key:
M means that the state data is modified by the process
U means that the state data is used bv the process but not modified

The use of a project schedule implies that there is a concept of time and the Engineering team, as well as outside organizations, understand and use this concept. It is a reasonable assumption that the Engineering team looks at watches and calendars regularly, and can relate times and dates to the project schedule.

5.5.2 Content

Much of the data appearing in this file is not determined by the Engineering team. For example: The Master Project Schedule and the Project Charter's formats are provided by the project sponsor who gives the two items to the Engineering team. The project schedule, which is merely a more detailed Master Project Schedule is normally pre-determined by management. The Engineering activity records need to appear in the form requested by the project sponsors or the organizations management.

The project schedule consists of a calendar and a number of task assignments. The project displays the tasks, reviews, milestones, etc., that occur on any specific date. Additionally, task assignments that are active on a specific date may be listed. Task assignments include the project, process (E0-E24), task, effort description, deliverables and special instructions. There are typically 20 + 5 * (number of increments) different processes for the project. The five results from the fact that there are five processes instantiated for each increment (which means those processes are identified by process name and increment number). The number of tasks based on each process is a management decision. A typical task assignment for a manual system is shown in Figure 5.5.2.1. Automated systems will provide different forms with the similar content.

Figure 5.5.2.1 Task Assignment

TASK ASSIGNMENT

Project: ______________________________
Process: ______________________________
Increment: ____________________________
Task: ____
____
____

Task Dates: From: __________ To: __________
Staff hours to be expended on task: ________________
Assigned to: ___________________________________
____
____
____

Deliverables: ___________________________________
____
____

Special Instructions: ____________________________
____
____
____
____

AD-A256 366 SOFTWARE TECHNOLOGY FOR ADAPTABLE RELIABLE SYSTEMS 3/4
(STARS) PROGRAM THE CL..(U) IBM FEDERAL SECTOR DIV
GAITHERSBURG MD R H COBB ET AL. 28 FEB 91 07001-001
UNCLASSIFIED XC-ESC F19628-88-D-0032 NL

Table 5.5.1 Project Management State Data Usage by Process

	Project Charter	Master Project Schedule	Engineering Activity Records	Project Schedule	Task Assignments	Metric:
E0: Cleanroom						U
E1: Project Invocation		M		M		U
E2: Program Management						U
E3: Project Information Management						U
E4: Software Solution Specification	U	U	U	U	U	U
E5: Software Development and Certification						U
E6: Prepare Final Project Releases	U	U	U	U	U	U
E7: Maintain Project Schedule		U	U	M	M	U
E8: Prepare For and Conduct Project Review	U	U	U	U	U	U
E9: Prepare and Submit Project Status Reports			M	U	U	M
E10: Receive Stimuli	M	M			M	U
E11: Submit a Question or Issue	U	U	U	U	U	U
E12: Understand Problem Domain				U	U	U
E13: Understand Solution Domain				U	U	U
E14: Write Specifications	U	U	U	U	U	U
E15: Write Construction Plan		U		U	U	U
E16: Increment Development						U
E17: Increment Certification						U
E18: Develop Increment i			M	U	U	U
E19: Develop Certification Tests for Increments 1..i				U	U	U
E20: Update Specifications			U	U	U	U
E21: Increase Understanding of Problem and Solution Domains as Required			M	U	U	U
E22: Build System with Increment j				U	U	U
E23: Certify Increments 1..j				U	U	U
E24: Correct Code Increments 1..j			M	U	U	U

Key:
M means that the state data is modified by the process
U means that the state data is used by the process but not modified

Figure 5.6.1 Issue/Question Form

Issue/Question Form

Name of Staff Member: ______________________ Date: __________
Project Name: __________ __________ I/Q Form Number: __________

Team: Specification / Development / Certification

Section 1: Question/Issue

Question/Issue needing resolution: ______________________________________
____
____
____
____
____

Check if additional question/issue materials are attached to this form: ____

Form submitted to: ______________________________

Section 2: Resolution

Name of Staff Member: ______________________ Date: ____________
Question/Issue resolved (check one): Orally: ____ Textually: ____

Briefly describe resolution (attach additional sheets as needed): ______________
____
____
____
____
____
____
____
____
____

Known ramifications of resolution: ______________________________________
____
____
____

Check if additional resolution materials are attached to this form: _______

Table 5.6.2 Questions or Issues State Data Usage by Process

	Questions or Issues	Metrics
E0: Cleanroom		
E1: Project Invocation	M	
E2: Program Management		
E3: Project Information Management		
E4: Software Solution Specification	U	U
E5: Software Development and Certification		
E6: Prepare Final Project Releases	U	U
E7: Maintain Project Schedule	U	
E8: Prepare For and Conduct Project Review	U	
E9: Prepare and Submit Project Status Reports	U	M
E10: Receive Stimuli	M	
E11: Submit a Question or Issue	M	
E12: Understand Problem Domain		
E13: Understand Solution Domain		
E14: Write Specifications		
E15: Write Construction Plan		
E16: Increment Development		
E17: Increment Certification		
E18: Develop Increment i		
E19: Develop Certification Tests for Increments 1..i		
E20: Update Specifications	U	
E21: Increase Understanding of Problem and Solution Domains as Required	U	
E22: Build System with Increment j		
E23: Certify Increments 1..j		
E24: Correct Code Increments 1..j		

Key:
M means that the state data is modified by the process
U means that the state data is used by the process but not modified

5.7.2 Change Control

When the source code for some increment is received by the Certification team from the Development team as a result of their completion of E18: Develop Increment i, the software as received is placed under configuration control. It is then compiled and any observed failures are recorded on Failure Reports by the Certification team. The Failure Report is returned to the Development team for correction. The Development team finds the reason for the failure and prepares an Engineering Change Notice (ECN) which records the reason for failure and defines the fix to the software which is intended to eliminate the failure. When the Development team is satisfied with the fix, the ECN is sent to the Certification team. Following acceptance of the ECN, the source code being maintained under configuration control is updated with the fix as supplied on the ECN. This process continues until the software successfully compiles.

Once the software successfully compiles the source code for the present increment, the compiled software for the present increment is assembled with the compiled software for prior increments in accordance with the assembly instructions received from the Development team. If failures are encountered, a Software Failure Report is prepared. The Software Failure Report is sent to Development team for resolution. The Development team records the fix in an ECN. Once the Development team is satisfied the fix is proper, the ECN is returned to the Certification team. The software and/or the assembly instructions are modified as instructed in the ECN. This process continues until the software is correctly assembled.

When the software is fully assembled, the Certification team commences measuring the reliability of the software by testing the software using the testing plan previously prepared. The Certification team records any observed failures on Software Failure Reports. When the Certification team feels so many failures have been observed that further testing has reached a point of diminishing returns, testing is suspended and failure reports are sent to the Development team for resolution. As the Development team finds fixes to remedy the observed failures, ECN's are prepared. Once the Development team is satisfied with the changes, the ECN's are returned to the Certification team. The Certification team then updates the software which is under configuration control. The process of compiling, assembly and testing is repeated. The process continues until the software is certified.

In addition to the Certification team finding problems, the Development team may also uncover difficulties or find an opportunity for an enhancement to the Pre-Release Software by reading or otherwise observing the code, or implementing a change in the requirements or specifications. In these cases, the Development team completes an ECN and submits it to a Configuration Control Board who makes a determination if the recommended change should be accepted or rejected. If the recommended change is accepted, the ECN and the modified code is forwarded to the Certification team for incorporation into the software and for testing.

The Certification team archives Failure Reports as they are prepared and Engineering Change Notices as they are received for analysis and tracking.

5.7.3 Content

Figure 9.4.4 defines a set of rules for code. These rules need to be tailored to each organization. Initially, an organization might want to start with these rules and eventually extend them to meet their special needs.

The use of and modification to the Pre-Release Software state data by each process is listed in Table 5.7.1.

5.8 Failure Reports and Engineering Changes

5.8.1 Organization and Procedures

Failure Reports and Engineering Changes should each have a separate form. These forms will be filed separately, but cross-referenced. The purpose of these files is to determine what changes have been and remain to be made to the code. Additionally, the extent of failures or changes will be understood, as that issue may affect the development. Finally, failure and change reports must be preserved to allow for reliability estimations, since the estimations are based on the number of engineering changes made. These forms may exist electronically or on paper.

All forms must be completed correctly and signed. Change control of the code is of the greatest importance (just as critical as change control of the specifications), extreme effort must be made to guarantee that the integrity of the controlled code library is preserved.

The process by which the Software Failure Reports and Engineering Change Notices work is described in Section 5.7.1, in the sub-section titled "Change Control."

Typical Software Failure Reports and Engineering Change Notices are shown in Figures 10.4.1 and 10.6.1, respectively.

The use of and modification to the Failure Reports and Engineering Changes state data by each process is listed in Table 5.8.1.

Table 5.7.1 Pre-Release Software State Data Usage by Process

		Software
E0:	Cleanroom	
E1:	Project Invocation	
E2:	Program Management	
E3:	Project Information Management	
E4:	Software Solution Specification	U
E5:	Software Development and Certification	
E6:	Prepare Final Project Releases	U
E7:	Maintain Project Schedule	U
E8:	Prepare For and Conduct Project Review	U
E9:	Prepare and Submit Project Status Reports	U
E10:	Receive Stimuli	
E11:	Submit a Question or Issue	
E12:	Understand Problem Domain	
E13:	Understand Solution Domain	
E14:	Write Specifications	
E15:	Write Construction Plan	
E16:	Increment Development	
E17:	Increment Certification	
E18:	Develop Increment i	
E19:	Develop Certification Tests for Increments 1..i	
E20:	Update Specifications	U
E21:	Increase Understanding of Problem and Solution Domains as Required	U
E22:	Build System with Increment j	M
E23:	Certify Increments 1..j	U
E24:	Correct Code Increments 1..j	U

Key:
M means that the state data is modified by the process
U means that the state data is used by the process but not modified

Table 5.8.1 Failure Reports and Engineering Changes State Data Usage by Process

	Failure Reports and Engineering Changes
E0: Cleanroom	
E1: Project Invocation	
E2: Program Management	
E3: Project Information Management	
E4: Software Solution Specification	U
E5: Software Development and Certification	
E6: Prepare Final Project Releases	U
E7: Maintain Project Schedule	U
E8: Prepare For and Conduct Project Review	U
E9: Prepare and Submit Project Status Reports	U
E10: Receive Stimuli	
E11: Submit a Question or Issue	
E12: Understand Problem Domain	
E13: Understand Solution Domain	
E14: Write Specifications	
E15: Write Construction Plan	
E16: Increment Development	
E17: Increment Certification	
E18: Develop Increment i	
E19: Develop Certification Tests for Increments 1..i	
E20: Update Specifications	U
E21: Increase Understanding of Problem and Solution Domains as Required	U
E22: Build System with Increment j	M
E23: Certify Increments 1..j	M
E24: Correct Code Increments 1..j	U

Key:
M means that the state data is modified by the process
U means that the state data is used by the process but not modified

Section 6
Responses For Cleanroom Engineering

In Section 3, thirteen responses to the software engineering process were defined as follows:

R1 Distribution Software
R2 Maintenance Software

The software as designed, implemented and certified as meeting the requirements can be regarded as the primary response of the Cleanroom Engineering project. This software is packaged in two forms. The first is the software ready for distribution — the Distribution Software. The second is the software ready for the organization that is responsible for archiving and maintaining the software — the Maintenance Software.

R3 Final Specification
R4 User Documentation
R5 Archived Documentation

The software must be documented in several forms as follows:

1. The Final Specification defines what the software does from both the user's standpoint and the computer's vantage point and the usage profile which the software has been designed and certified to satisfy.

2. The User Documentation which is employed by users to help utilize the software. The format and extent of User Documentation depends on the requirements. Typically, it may include reference manuals, tutorial manuals, education material, marketing documentation.

3. The Archived Documentation represents the final **state** of the Cleanroom Engineering development process when the software is complete and certified. From one point of view, software development can be regarded as developing the **state** during the entire software solution specification, development and certification process; and once the software is complete, the converting of the **state** into the responses as defined above. The Archived Documentation is a permanent set of documentation that represents the final **state**. The extent and formality of the Archived Documentation is specified by the requirements.

R6 Project Review Documentation

Project reviews are meetings between the Engineering team and one or more outside organizations. Engineering teams prepare documentation for use at project reviews. Project reviews are typically held for projects at critical points in the life cycle. In this way project sponsors and/or management can assess progress and the degree to which the software as it is being designed meets the needs which are to be satisfied by the software. Additionally, project reviews can be called by project management when they see a need, or on a schedule, such as a monthly status review. The exact nature of the documentation that is produced for these reviews typically depends on the nature of the development contract or the customs of the organization. For example, if the contract is being developed under a government contract, the reviews may be specified by 2167 or 2167A. Other organizations have their own specifications for project review documentation. This project review documentation must be produced at the specified time in the specified format from the state data. This documentation is produced at many different points in the project life cycle. In many cases, the development organization is free to propose a format for Project Review Documentation.

R7 Project Schedule

The Project Master Schedule is updated with schedule information received from the Cleanroom Engineering Team. The Project Schedule is the response which provides this information. It is a published description of the schedule of milestones and deliverables between the Engineering teams and between the Engineering team and management. The dates given in the Project Schedule commit the Engineering team to a schedule for production of the system, to which "management" can hold the Engineering team accountable.

R8 Management Metrics

Measurements of engineering processes are collected routinely. The measurements are used by management to control the development and to study development performance to find improvements for future projects. Individual managements will want to collect different information; therefore, the exact metrics collected will differ from project to project. These measurements as defined for permanent archiving are referred to as Management Metrics.

R9 Project Status Reports

Project Status Reports are distributed to management based on the schedule dictated in the Master Project Schedule. Information in the reports allows management to assess the status of the project in regards to budget and resources and make modifications to the project as necessary.

R10 Externally Submitted Questions or Issues

During a project, there are often questions or issues that cannot be answered by members of the Certification, Development or Specification teams. These questions or issues are submitted to organizations outside the engineering task in order to be resolved. The issues are resolved, with their response being either Documents/Working Papers or Informal Communication.

R11 Schedule Change Request

During a project, the Engineering team may determine that a milestone or deliverable may not be completed on schedule. For these types of situations, the Engineering team must have the opportunity to request a schedule modification. This request will prompt management to analyze the issue and determine whether to grant a schedule change. If the change is accepted, the Engineering team will receive a new Master Project Schedule as a stimulus.

R12 Suspend Project Pending Management Decision

During a project, the case may arise where the Engineering team believes the project must be suspended; for example, upon determining that the code does not meet the specifications, or that the specifications do not meet the requirements. Since management has the final decision as to whether the project should be suspended, this response is a prompt to management to closely assess the project and make a decision as to how the project will continue or whether the project should be reorganized or cancelled.

R13 Project Reports

During a project, the Specification, Development or Certification teams may produce a document that has use outside of the engineering team. These documents are published outside the engineering team, so other individuals in the organization can use the knowledge gained by those members of the Engineering team.

In some cases, a project sponsor will decide that some other responses are necessary, in addition to the ones that appear in this section. These responses are not addressed because it would be impossible to identify the full range of potential responses. The responses appearing below are a fundamental set of project responses. Some additional responses can be accounted for by viewing them as material that must be a part of a review. As reviewed material, it will be delivered to the sponsor. For other responses, it may be necessary to tailor this document for the specific organization and project in order to account for such requests.

Due to the nature of the state data invented for the process and the nature of Cleanroom Engineering practices once the project is complete it is very easy to prepare all project responses.

Nine of these responses (R1, R2, R3, R4, R5, R6, R7, R8 and R13) are directly or indirectly derived from state data. Some of these responses will be reformatted to respond to organization and user requirements and others will just be duplicates of the final state.

R1 Distribution Software — Since one of the first things the Certification team must do with the software is install the software in the target environment for execution, they have been assembling the software from distribution format since executing the very first increment. Therefore, the distribution software is directly available in state.

R2 Maintenance Software — The software has been maintained during the entire development cycle so again the maintenance software is directly available in state.

R3 Software Specifications — Directly available in state. The specifications have been repeatedly published in distribution form so they are available both in printed and electronic formats.

R4 Archived Documentation — This response is an archived copy of the five engineering state data files so they are available for future reference.

R5 User Documentation — Very often it is necessary to provide documentation in forms other than a User's Reference Manual. In those cases these other forms of documentation need to be prepared from the specifications and the authors of these documents must show they are equivalent to the specifications. Once they are written, they will be available. It is not possible to define in advance how such documentation will be formatted because that will differ for each project depending on the target audience for the software.

R6 Project Review Documentation The format of these documents will depend on the wishes of the project.sponsor and the review team. But, as discussed in Section 3.10, this review documentation must be obtained directly and entirely from the state.

R7 Project Schedule Project schedules are contained in the state. The format in which someone may want schedule information will vary. The process used to maintain the project schedule should be capable of meeting the mix of report formats that people will desire.

R8 Management Metrics The organization will have defined what management metrics are to be collected so all metrics which will be available are in state. Provided a good data base system is selected for storing metrics information formatting, the information to satisfy individual requests will not be difficult. The data base capabilities defined for each project should be compatible with organization-wide standards so it will be easy to archive the metrics for future analysis. If this is not the case, it will take some effort to reformat the data.

R13 Project Reports The project reports are written from information found in state data files and are also stored in state data. The format of the paper in state data is the same as its form as a response. Working papers can also be converted into project reports and published as such.

The other responses are simple paper forms as follows:

R9 Project Status Report Most organizations have a form in which status reports are to be submitted. Status reports are typically memos written at a predefined interval to indicate progress against goals, problems encountered and other matters of interest to the organization.

R10 External Issues/Questions Forms that can be used for this purpose were defined in Section 5.6.

R11 Schedule Change Request The format for a schedule change request is installation dependent. Most likely it will be a memo explaining the change requested along with a rationale.

R12 Suspend Project This is a crisis response and will most likely be delivered in person in a very unpleasant meeting. The form of the notice will be very dependent on the specific manager and the situation.

Section 7
Managing A Cleanroom Project

In this section the Cleanroom Project Cycle is examined from the vantage point of the engineering manager. In the Cleanroom environment the engineering manager has a-well defined mission which can be stated as follows:

> Given the responsibility to specify, design and certify software which is to fulfill a given mission the engineering manager must organize, plan, direct, measure and control the activities of the three Cleanroom engineering teams so these teams develop high quality software (i.e., the software exhibits few, if any, failures in use) with high productivity.

The primary purpose of this section is to develop a list of the issues that a manager needs to consider relative to organizing, planning, directing, measuring and controlling a Cleanroom project.

The following points will be discussed in greater detail in the following sections:

7.1 Productivity

7.2 Process Models and Productivity

7.3 Process Models and Management

7.4 Process Improvement

7.1 Productivity

A software manager must define and produce software that fulfills its assigned mission with the greatest possible efficiency. Producing software that does its assigned job should be regarded as a constraint. The manager who can perform the job more efficiently is the better manager. Since designing and producing software is a people-intensive process, the main efficiency driver is productivity.

As software managers perform software projects, they need to be concerned with three sources of low productivity: effort, realization and accuracy.

> **Effort productivity** is involved with the type of things many of us consider when we think of productivity. That is how hard people work. Effort productivity is often associated with imagining managers using threats and worse to establish a high degree of effort productivity. Since software development engineers are professionals, managers do not need to do much to establish effort productivity. The main thing managers can do to establish effort productivity is to provide good communications combined with a fair reward system.
>
> Management involves human interaction. It is necessary that both the manager and each engineer understand each other. It is also vital that engineers be able to communicate with each other. More problems are caused by a lack of common understanding of what is being said. This is especially true when both parties to the communication do not have a common understanding of what should be

going on. The best way to solve communication problems is to establish a common basis for communication and understanding. Activities that have a well understood process model have many less communication problems than organizations that try to operate without a well understood process model. Therefore, one important role of a process model is to provide a common basis for communications.

A prerequisite to having a good reward system is to give people the ability to perform what they are being asked to do. For example, software engineers are being asked to produce reliable software without being provided with the tools to perform. This gap makes it nearly impossible to put in place a fair reward system that works. Cleanroom directly attacks this problem.

Realization productivity is associated with having all the tools and other things required to perform the task at hand. Each time someone must wait for something, is working with the wrong thing, is working from a guess, does not have the right tool, etc., productivity is suffering. In a professional organization effort, productivity is the responsibility of the individual professional and realization productivity is the responsibility of the manager. In software development individual engineers typically exhibit a high degree of effort productivity while managers typically are not providing an environment that permits engineers to achieve a high degree of realization productivity. The reason for this is that trial-and-error software development does not lend itself to organizing to minimize realization loss since the process is out of intellectual control. Cleanroom directly attacks this problem.

Accuracy productivity is associated with getting it right the first time. Whenever original work must be reworked, productivity suffers. In environments where there is a high degree of rework, productivity is low. In software development there is typically a high degree of rework. First, engineers attempt to verify code by testing which leads to rework. In software development attempting to test in quality is the norm and this leads to extensive rework. In software development lack of accuracy is the major cause of low productivity. Cleanroom directly attacks this cause of productivity shortfalls by providing individual engineers with mental processes that permit them to perform their work with a high degree probability of getting the work right so rework is avoided.

The Cleanroom process model as developed in the previous sections permits managers to increase productivity in all three areas. Effort productivity increases due to the team style of working. Realization productivity increases for two reasons. First, because Cleanroom enables organizations to replace heuristic, undefined, individualistic practices by rigorous, team-oriented, mathematically derived practices that give engineers complete guidance of how to go about their inventive process. Second, it gives managers guidance on how to organize and to plan and control projects so the engineers can maintain a high degree of realization productivity. And the Cleanroom practices provide engineers with guidance on how to develop software that requires an extraordinary low rate of rework based on current standards.

7.2 Process Models and Productivity

Process models are a prerequisite to establishing an environment in which engineers and managers can communicate and achieve any reasonable level of productivity. A prerequisite to establishing a process model is that the methods of converting inputs to outputs for the process must be well understood. All major engineering disciplines except software have well established and widely understood process models.

Software development managers have had a great deal of difficulty developing a useful process model for software development because the methods of converting inputs to outputs are not well understood. Cleanroom practices establish well defined methods for converting inputs to outputs when developing software. Therefore, it is possible to establish a useful process model for software development when Cleanroom practices are being used. The reasons it is possible for organizations and managers to clearly organize for software development when using Cleanroom and not when they are using heuristic, trial and errors practices can be summarized as follows:

> Cleanroom clearly defines what has to be done, both on a general level (processes and control flow among the processes) and on a detailed level (actual engineering tasks).
>
> Cleanroom clearly defines how to accurately measure the design process.
>
> Cleanroom emphasizes a team-oriented approach.
>
> Cleanroom establishes a separation of responsibilities and concerns, and clearly describes the activities done by each team.
>
> Cleanroom dictates that an incremental development and certification approach be followed, allowing developers and certifiers to maintain intellectual control over the system they are developing.

7.3 Process Models And Management

A manager of a software development organization uses the software development process model to guide each of the five major management activities. The following sections discuss each of these aspects for the simpler half of the software development cycle. That is going from specifications to certified software ready for deployment. The process of going from specifications to certified software is the easier of the two software development problems. The more difficult problem, converting dream or concept into specifications is not addressed in the sections below. These five activities are not sequential as there is overlap and they are performed in a repetitive loop for the entire course of the project.

7.3.1 Organizing In A Cleanroom Environment

The goal of a manager, while organizing for a project, is to prepare a situation, which will lead to the highest probability that the project will succeed where success is measured in terms of customer satisfaction and minimum cost. Minimum cost is achieved by organizing to achieve high productivity, and customer satisfaction is obtained by defining a product that fulfills the assigned mission with high reliability (i.e., no (few) failures). Ideally, the engineering team manager wants to consider and if possible, solve, all foreseeable issues. Before any software project begins, a number of issues related to the project must be organized. These include at least the following: process manual, process management system, engineering handbooks, computer hardware, software tools, state data storage, training, team formation, office facilities and communications. Each of these issues are addressed below:

> A clear **process manual** is critical for a successful project. The process manual justifies the process to be used, and describes it. It presents a process user's guide for project managers and clearly

describes what the rest of the engineering staff is to do. Clearly defined processes, tasks and completion criteria must be defined to allow the project manager to direct, measure and control the project. The processes, engineering tasks and completion criteria need to be made publicly available to the rest of the engineering team to provide for a clear basis of communications. The Cleanroom process model as developed in this document is intended to serve as a basis for specialization for software development projects. It can then be refined based on actual use on software development projects, thereby facilitating process improvement.

A **Process Management System** is very helpful to a manager in implementing, communicating and managing the selected process for the project. Researchers are just beginning to develop Process Management Systems dedicated toward software development. For example, the STARS Breakthrough initiative (IR23) is supporting a prototyping effort to develop a process management system.

Engineering Handbooks are extremely important for those participating in a project. The process manual is a process user's guide for project managers and engineers. Engineering handbooks describe, in detail, how a specific task or set of tasks are to be completed in a form that makes it easy for a practicing engineer to recall specific facts when required. This is as opposed to text books which are designed to teach the facts and methods in the first place. The task descriptions in an Engineering Handbook present step-by-step processes which, when followed, lead to the completion of a task. Conditions for the completion of each task are also given. With the combination of process manuals and handbooks, all members of the engineering team for a software project have detailed descriptions of the processes they must complete. Engineering handbooks are not yet available for software engineering since trial-and-error development is not conducive to the rigor required of an engineering handbook. It is now possible to develop engineering handbooks for Cleanroom Engineering. Until the handbooks are complete, engineers practicing Cleanroom must use textbooks, papers from professional journals or lecture notes from Cleanroom Engineering courses.

The engineering team requires **computer hardware** to perform work. All Cleanroom Engineers require a personal computer or low-end workstation. These can be PC's (hopefully with Windows), Macintoshes, or low-end UNIX workstations. It is desirable that the PC's and/or workstations be linked with communications - a LAN for people working in the same location and a wide area network for people working over a wide network. The Certification team needs access to the target machine in order to test the software. It has been found that Cleanroom Engineers do not need powerful workstations with sophisticated software to perform efficiently. This feature alone provides organizations the potential for great savings. Consider that today most software developers require a large engineering workstation to support the compiler. Compilers like Ada require a lot of computing power and are expensive. When using Ada it is common for organizations to pay as much as $15,000 for an engineering workstation with Ada and other software. If this is a large DoD project, a hundred engineers may be involved. The total investment would then be $1,500,000. With Cleanroom, there would be a savings because the same work could be done in less time with fewer people; but to make the point about hardware, assume the same hundred people are required. Workstations with software in the Cleanroom environment could be acquired for less than $3,500. The Certification team may need two engineering workstations so the total cost may be $380,000. A significant savings for a better, more productive environment.

In addition to the hardware, the **computer software** required must be selected and acquired. Today, Cleanroom Engineers use word processing software. This is sufficient for design and coding, as well as for specification and other writing requirements. Certification team members need access to a compiler and configuration management software. In the future specific software to automate some labor intensive portions of Cleanroom will become available. Cleanroom tools are not yet available because until now effort has been concentrated on developing and refining the theory and engineering practices. Now that the theory is essentially complete, it is time to start developing Cleanroom productivity aids.

A means for **storing state data** must also be put in place. The key issues in selecting the means of storage for the state data are: 1) to ensure that the information is properly preserved, and 2) to allow state data to be easily accessed and used by those who need it. A corollary of the second point is that the information should not be readily available to those who do not have need for that part of the state data. The state data can, and often will, exist in a variety of forms. Some may appear in notebooks or be stored in file cabinets, while other information may be stored electronically. As long as the means of storage preserves the information safely and accurately, and is easy and efficient to find and use, that means is acceptable. It is critical to note that the means of storage will differ according to the hardware and software resources available, but that does not change the organization of the state data. For example, state data stored on paper and on floppy disks (in a PC based system) is organized in the same way as the same state data appearing strictly on-line (in a server-based system). Accessing the state data may be more or less convenient, but the considerations as to how material is modified or how it is kept together are the same, whatever the means of storage. The following questions can help define the best arrangement of state data storage for each project:

a) How will access to state data be controlled?
b) How will additions of state data be controlled?
c) How will deletions of state data be controlled?
d) How will modifications to state data be controlled?
e) How will state data be stored in a convenient format given the size and dispersion of the project? For example, it may be more convenient that each category of each state data file be kept in separate files (physical or electronic).
f) How will the integrity of the state data be checked?
g) What are the means of ensuring
   - that the people who should know about new or updated state data know about it, and
   - that the people who need to know have actually looked at the necessary state data.

The solution to these state data considerations will depend on the size and geographical dispersion of the teams. In smaller projects the best arrangement may be manual files backed up by disk files. In larger projects more elaborate online files are normally necessary.

Engineering team members need to be **trained**. Although the Cleanroom technologies exhibit only evolutionary technological differences, these technologies must be well understood before the Cleanroom project commences. Training, for those who will be on the Development team, in designing systems using Box Structures and implementing systems using stepwise refinement and functional verification is necessary. Those who will be a part of the Certification team must know how to engineer software under statistical quality control. The training must include lectures and workshops, in order to help the engineering team staff develop the necessary skills.

Team formation. The engineering team manager must form the Specification, Development and Certification teams. If the project is large, there may be more than one of each team. If the project is small, an individual may be a part of more than one team. Each member of a specific team should have the necessary training and skills for that team. The ratios of staff members between teams must be determined inside an organization. Typically, there will be more developers than certifiers or specifiers. Whether there are twice as many or just one more developer than certifier must be determined in the environment, based on the staff and the problem at hand. In terms of a general trait, a manager for a Cleanroom team wants the staff to be people who work well with others, as Cleanroom emphasizes a team approach to development. Generally, there will be three or four members per Development team, and one or two per Certification team. These team sizes permit at least four increments to be completed and certified by a single team in a year except in the most unusual circumstances.

The **office and conference facilities** must be organized. Individuals must have sufficient working space. They also need easy access to the computers and state data that they need. Meeting rooms are also necessary for team reviews, meetings, etc. These rooms only need to be large enough to comfortably hold a Development, Specification or Certification team.' White boards are a useful feature in the meeting rooms since they are a useful medium for the explanation of concepts during proof reviews. There is a need for access to a large room for project reviews.

Finally, **communication protocols** must be specified while organizing for the project. This includes communication between the Specification, Development and Certification team and communication between one or all of the teams and outside organizations, such as the customer. Intra-team communications should be oral and written, especially when decisions are made. In general, the engineering team manager should be informed of or should conduct all external communications. External communications should appear in a written form as much as possible. All communications (external or internal) that affect more than one person should be written down and distributed (maybe with a distribution list), in order to ensure that everyone who needs to learn about something will have the opportunity.

In addressing these ten issues, the engineering team manager will have made significant progress in organizing the process for success. It is a good practice to prepare a formal project organization document that addresses these ten points and defines what is to be done in each case. There may be unforeseen issues, before or during the project, that may lead to the need for a re-organization. If that occurs, the original resolutions of the ten issues may need to be modified to address the new situation. Real time organizing is more difficult but will address the same issues that the original organizing session addressed.

Organizing for a project may be the most important task a manager can perform. If the manager does a good job of organizing, the project is likely to go smoothly even if the manager falls down on the other aspects of management. On the other hand, if a manager does a poor job of organizing, no matter how well all the aspects of management are performed, the project is likely to be a disaster.

7.3.2 Planning In A Cleanroom Environment

The goal of the planning phase is to develop a realization of a process for a specific project. This entails significant effort in understanding and analyzing the process and the project, and the packaging of that understanding and analysis as the project plan. Specifically, the engineering team manager must assess the project parameters and the Cleanroom parameters, thus understanding the project and process. This information can then be synthesized into a Cleanroom project plan. Finally, the plan must be documented in order to facilitate communications.

In order to understand the project, two important parameters must be addressed. First, the incremental approach to development and certification must be clarified. Although this issue is process dependent, it primarily focuses on the project. The number of increments, their sizes and their relative complexity must be understood. Additionally, the number of reviews and test scenarios (which are a function of the desired reliability) must be estimated. The organization of the increments is also dependent on the number and size of the Development and Certification teams. Also, the amount of remaining bottom-up exploration must be estimated. This is the amount of time spent understanding the problem and solution domain before the actual process of forming a solution begins.

Second, the resource estimation for the project must be done. The manager must determine the staff and computer resources required for each subsystem or increment. This analysis leads to the best possible delegation of resources to the project.

To estimate resource requirements the production factors for the project must be determined. A typical production factor is an estimate of how much code and other items can be created or completed per unit time, using the Cleanroom process. Production factors are somewhat project dependent, but primarily concentrate on the process being used. Specifically, development and certification productivity must be understood. The staff productivity of both teams are dependent on a number of common issues. In terms of staffing, size and available hours, as well as the extent of training, must be understood. Both of these factors are specified during the organizing activity and lead to a staff productivity estimate. In addition, the experience that the staff has, both in the project domain and with the process, must be determined. Productivity is also dependent on the resources available for the project, both hardware and software (such as tools, etc.). These factors were determined during the organizing activity. All of these factors yield a realization productivity estimate. Finally, the project manager must estimate how much of the project may need to be redone. This includes changes to requirements, as well as the probability that one of the teams will make a mistake, such as leaving too many errors in the code. For this, an accuracy productivity estimate must also be made. Once the information listed above is available, the project manager can estimate developer and certifier productivity, based on the effort, realization and accuracy productivity factors. The developer productivity is a rate at which code or other development items can be produced during the project. The certification productivity is the rate at which test scenarios can be created or executed during the project.

Another very important factor that influences planned productivity is the amount of software that is planned to be reused. Software that is planned for reuse may have to be certified to insure it provides the required reliability. This can be done by developing a usage profile and constructing usage tests. The reuse plan will affect the order of construction. Planning for reuse in the Cleanroom environment was one subject of the IR 70 Phase 1 report.

Finally, the Cleanroom model must be understood. The model can be combined with project-specific knowledge to permit the engineering team manager to tailor the Cleanroom process model in order to best complete the project.

This process model is meant to be a description of the Cleanroom process, not necessarily a prescription of how to complete the Cleanroom process in all situations. The process model is not used directly to allocate effort to and then control the performance of a project. The process of using the model to plan for, to schedule and then control a project requires three steps as follows:

1. The process model is used to define a list of all the processes required for the project of interest (the project processes). To prepare the list requires knowledge of the specific project. For example, number of increments, the number of reviews and when they are to occur, etc.

2. The process model is used to determine the precedence relationships between the project processes. The result is a precedence diagram, commonly called a PERT chart. Given project knowledge it is possible to assign the effort required to perform each of the project processes. Therefore, it is possible to determine the critical path through the precedence diagram. The result is the minimum project duration.

3. Then, given resources to perform the project, it is possible to allocate the project processes to resources and develop a schedule. There are many possible schedules that can be used to perform the project. Project managers, of course, try to find the best schedule. Best is defined according to many factors, including risk minimization.

Tailoring is not easy and will often continue past the planning stage and into the project.

Upon the understanding of the project and process parameters, the engineering team manager records the project plan. In addition to creating the plan, the justifications for the plan must be documented. This documentation will be used to communicate to and persuade upper management that the plan is the right one. The plan should be complete and consistent in defining the goal, process and strategy, in order to minimize ambiguities and to maximize precision. That will make the plan easier to implement.

The need may arise to replan the project after it has commenced. Unfortunately, this often occurs due to situations outside of the engineering team manager's control. If this situation occurs, the project and process must be re-assessed in light of the new situation, and the new plan must be redone accordingly. The justifications for the original plan will be useful again at this stage, not only for its original purpose of presentation to upper management, but also to minimize rework by outlining the strategy that led to the original plan. That strategy may still be correct, or may leave the staff at a more advanced point, requiring less rework than starting at the beginning of the planning process. Ideally, the need for replanning will not occur, or will only be minor. Unfortunately, that is not always the case; and it behooves the manager to leave as detailed a plan creation trail as possible.

7.3.3 Directing In A Cleanroom Environment

Once the project has commenced, the process of directing (that is, ensuring that the plan is followed) is critical to the success of the project. The plan is issued at the commencement of the project, Project Invocation (E0), and the activities of the engineering team commence, based on the plan. The plan is a detailed and tailored version of the Cleanroom process described in Sections 4 and 5 and will clearly define the processes that the Specification, Development and Certification teams will follow. Additionally, the flow of control between processes is well defined, in other words, the temporal relationships between all of the processes are well defined. The manner by which a manager can ensure that the process is being correctly followed is by checking the completion conditions at the end of each process. The completion conditions are known to all of the engineering staff in the Cleanroom process, since they are a part of the plan. These conditions are checked by members of the Specification, Development or Certification teams when they believe that the specific process is completed, and also validated by the manager to confirm that the process is completed. It is a good idea to prepare check lists of completion criteria for each process in the project process hierarchy. These check lists should be distributed to the teams with their assignments. The team can then sign off that all completion conditions have been satisfied when they are complete. In this way everyone knows what is expected and what has been accomplished. Managers should validate that the defined Cleanroom practices are being followed by sampling the performance of teams on a random basis. This insures maximum conformance with minimum expenditure of effort. Cleanroom completion conditions checklists are included in Appendix A.

In addition, the description of the process (the manner in which the process is followed) also facilitates how it is directed. The team-oriented, public engineering process obligates engineering team members to audit each other's work, minimizing the potential of major mistakes. Developers and certifiers each use a common language (developers use BDL and PDL, while certifiers have clearly defined usage profiles and mathematical processes). The use of a common language helps minimize assumptions and ambiguities, and enhances the chances for completeness and correctness. As developers must depend on each other, not on compilers or their own test drivers, they tend to be more careful, leading to better results.

The public nature of the state data, another by-product of the process also eases the task of directing the process. Checking the preserved documentation periodically will give the project manager a clear understanding of how well the process is being followed and where additional direction is needed.

The engineering team must also be given the proper incentives and disincentives to follow the plan correctly. The set of incentives and disincentives for a Cleanroom project will differ from those for a typical project. For example, rewarding the developers by the amount of code they write is not a proper incentive while rewarding the lack of failures in code is a good incentive. Typically, developers are incentivized to spend a significant amount of time doing structural testing. This activity should be strongly disincentivized with Cleanroom. With typical processes, the need to keep the completeness and consistency of all design materials is not incentivized, since it does cost money. With Cleanroom, this activity should be strongly incentivized as it saves money. A proper set of incentives and disincentives will make the engineering team more motivated to follow the plan, making the project easier to direct.

During the course of the project, the project manager will make real time observations and, as a result, modify the schedule to account for difficulties or to take advantage of opportunities. Examples include:

> A process is scheduled to begin and, for any number of reasons, it is determined the process should not start as scheduled. As a result the project manager does not issue the instructions to invoke the process.
>
> A process is in progress and, either because of events observed in the performance of that process or in some other process, the manger decides to interrupt performing on that task.

7.3.4 Measuring In A Cleanroom Environment

Measuring, in a management sense, are the activities involved in understanding the actual state of the project, in order to make decisions on how the process needs to be modified. Measuring, in this sense, can be both quantitative and qualitative. It is the sum of all observations and other stimuli that the engineering team manager receives during a project. The ability to measure, and to acquire useful measurements, comes as a result of the Cleanroom process. The ways in which a manager can measure a project can be put under three broad categories: reviews, metrics and personal interaction.

In the Cleanroom process, there are two levels of reviews, project reviews and team reviews. The project reviews are detailed in Section 4.8, in process E8, which is called "Prepare For and Conduct Project Review." During the review, the project manager has the opportunity to hear opinions on the status of the project from a variety of different perspectives. These perspectives not only include the Specification, Development and Certification teams, but also the client and other outside organizations. Each of these perspectives will give the project manager some insight as to the state of the project at present.

The team reviews also give the project manager an understanding of the state of the project, although at a more detailed level. The Specification team reviews the specifications and changes that may need to be made to them. This is done in the Write Specification process (E14) and the Update Specification process (E20). The Development team reviews the proof conversations, at both the design and code level. These reviews are primarily done in the Develop Increment i process (E18), and also done some in the Correct Code Increment 1..j (E24). Finally, the Certification team looks at the various testing artifacts in their reviews, ensuring that their certification approach will follow the usage profile of the target software. The reviews are fairly limited to the Develop Certification Tests for Increments 1..i process (E19). A manager who participates in review, or more often receives the results of the reviews, has a relatively accurate assessment of the state of that part of the project. The manager knows that the reviews are a result of the effort of the entire team, with a limited set of concerns. For example, a certain Development team is only concerned with the design and coding of one relatively small increment during a certain period of time. The results should be quite dependable. These review points are a natural and required part of the Cleanroom process, they are not viewed as management assessment points.

Measuring is also done by gathering quantitative metrics from the Cleanroom process. These metrics are generally a natural by-product of the Cleanroom process, making them easy to gather. These metrics are described, in detail, in Section 4.9, in the Prepare and Submit Project Status Reports process (E9). The System Reliability measurement, for example, gives a clear and accurate assessment of how the product will work in the field. They, along with other measures, will point towards any necessary changes that may need to be made to the process. The metrics are gathered during all 25 processes to directly reflect all aspects of the Cleanroom model.

During a project, the manager will interact with members of the engineering team staff. These interactions may occur as meetings or in one-on-one situations. Knowledge gained from these types of activities will also assist the manager in understanding the present state of the project. Additionally, the manager is continually assessing the status of the project by determining which completion conditions for processes have been met. At the end of each process, the manager will determine whether a process is finished by checking that the completion conditions are fulfilled. These conditions are identical to those generated by the engineering team for each process, so there should be no surprises to the manager or the engineering team since both had to look at the same checklist. The checklists for the 25 processes in Cleanroom appear in Appendix A. For each of these checklists, a completion condition is dated when the manager determines that it has been fulfilled. Engineering team members do the same, and everyone signs the form. In that way, accountability for decisions is clearly established by having a person available to clarify the resolution of any completion condition. The checklists can clearly describe the state of the processes in a project.

The measures that the manager receives during the process (as a result of the reviews, the quantitative metrics) and management interaction lead to an accurate portrayal of the current state of the project. This understanding of the actual state of the project gives the project manager more complete information for the decisions that may need to be made to improve the process.

7.3.5 Controlling In A Cleanroom Environment

The controlling activity includes the comparison of the actual project to the plan, and the actions that are a result of the comparison. The comparison itself is done by looking at the results of the measuring process and at the plan, and noting any differences. The differences may be major or minor, as well as many or few.

For the differences found, the project manager must decide what actions to take. Of course, a reasonable and very realistic action is the decision not to take any action. In this situation, the project continues exactly as planned, with no modifications whatsoever. If an action is to be taken, the manager has a number of options. The manager may decide that sufficient information is unavailable for an action to be taken. In that case, the measuring activity will be modified to allow for more complete information to be found. In another case, the manager may determine that the plan is correct and that one or all of the teams have not been following it. In that case, the manager will redirect the engineering team by clarifying the plan, or by issuing new incentives or disincentives. If the manager decides that the plan needs to be change, a replanning activity may be required. The replan will result in a new Cleanroom project plan. It will appear after a thorough analysis of the project and the Cleanroom parameters. Finally, in the worst case, the manager may decide to reorganize the project. When this occurs, the process manuals, handbooks, hardware, software, state data storage, training teams, facilities and/or communications may need to be changed. This level of action will probably result in a new project and should be viewed as a last resort.

Each possible level of action will return the project manager to one of the previous stages of the management process. The actions also may also affect the engineering team and the Cleanroom processes they are completing. The actions may change a part of a process, an entire process, or the entire set of processes (i.e., the project). The iterative process that the project manager follows will eventually (hopefully) end with the manager determining that the project is completed, all obligations are fulfilled, and the action is to start the next project.

7.4 Process Improvement

The preceding discussion has concentrated on managing a single project or managing in the small. Managers also need to manage in the large by constantly improving the means by which they conduct projects. A major component of managing in the large is process improvement. The major thrust of process improvement is to observe what occurred on a project and learn what worked well and what did not work well. These observations are used to implement activities directed at improving some aspect of the process or other resources used to support software development projects. All managers use a Process Improvement Paradigm to improve how they are organized to perform projects. More successful managers perform process improvement better than less successful managers.

The way process improvement works is that data (metrics) are collected about how well the processes and other resources worked on the current project. These observations are analyzed and correlated with measurements on other projects to develop lessons learned which are then used to implement a better set of resources for the next project. The results may be better or worse. The result of this process is a whole series of results from natural experiments which the manager analyzes and uses to constantly improve the resources assigned to software development. This process improvement paradigm is not unique to software. In fact, the process improvement paradigm is much more mature in most (maybe all) other engineering fields.

Section 8
Cleanroom Specification Team Practices

In this section the Cleanroom Project cycle is examined from the vantage point of the Specification team. In the Cleanroom environment, the Specification team has a well-defined mission which is stated as follows:

> Given a general set of requirements for a software system, define a specific function to meet those requirements, which maps any possible sequence of user stimuli to the next correct response and a real time requirement for returning that correct response; further decompose this function into a nested set of increments that accumulate into the total function; further define the statistical usage of the software by mapping every possible sequence of stimuli to a probability distribution for the next stimulus.

In this short statement, the term "function" may be generalized to the term "relation" to admit more than one possible correct response for a given sequence of stimuli. Note that the specification is a Black Box that makes no reference to stored data, only to sequences of stimuli from which data can be captured and stored in a specific system solution.

In performing this mission the Specification team is performing the engineering tasks defined in E4: Software Solution Specification, which is made up of

E12: Understand Problem Domain
E13: Understand Solution Domain
E14: Write Specifications
E15: Write Construction Plan,

in Section 4, E20: Update Specifications and E21: Increase Understanding of Problem and Solution Domains as Required.

The Specification team also contributes to performing the work defined in E2: Program Management, E3: Project Information Management, and E6: Prepare Final Project Releases.

The following points will be discussed in greater detail in the following sections:

1. Specifications and Construction Planning
2. The Specification Documents
3. Recording The Specification
4. Developing The Expected Usage Profile

8.1 Specifications and Construction Planning

Precision in specifications requires formal languages, just as programming does. A formal specification defines not only legal system stimuli, but legal stimuli sequences; and for each legal stimuli sequence, a set of one or more legal responses. Any such formal specification, in any language, is a mathematical relation - a set of ordered pairs whose first members are stimuli sequences and second members a very important meeting. The form of the notice will be very dependent on the specific manager and the situation.

are responses. Then, there is a very direct and simple mathematical definition for a program meeting a specification. It is that the program determines a correct response for every stimuli sequence in the domain of the specification.

In Cleanroom Engineering, specifications are extended in two separate ways to create a structured software development. First, the formal specifications are structured into a set of nested sub-specifications, each a strict subset of the preceding sub-specification. Then, beginning with the smallest sub-specification, a pipeline of software increments is defined with each step going to the next larger sub-specification. Second, the usage of the specified system is defined as a statistical distribution over all possible stimuli sequences. The structured Cleanroom process makes statistical quality control possible in subsequent incremental software development to the specifications. The usage statistics provides a statistical basis for testing and certification of the reliability of the software in meeting its specifications.

A structured software development defines not only what a software system is to be when finished, but also a construction plan to design and test the software in a pipeline of subsystems, step by step. The pipeline must define step sizes that the development team can complete without debugging prior to delivery to the Certification team. Well educated and disciplined Development teams may handle step sizes up to ten thousand lines of high-level code or more. But the structured design must also determine a satisfactory set of user executable increments for the pipeline of overlapping development and test operations. The pipeline must also define the statistical usage of each partial subsystem for the testing and certification process.

The first task of a Specification team is prepare and publish a specification for the software. As noted above, the specification should be in six parts

Mission Volume
User's Reference Manual Volume
Functional (Black Box) Specification Volume
Functional Specification Verification Volume
Expected User Profile Volume
Construction Plan Volume

The Mission Volume defines the mission the software is to perform in terms of function and performance at an executive level. The User's Reference Manual Volume defines the precise stimuli users can enter and the precise responses users can observe, organized in a form to permit effective user referencing during operations. The Function (Black Box) Specification Volume provides the Black Box response to every stimulus based on the preceding stimuli sequence and the real time performance required for each stimulus. The Functional Specification Verification Volume presents the justification that the Function (Black Box) Specification is correct as written. The Expected Usage Profile Volume defines the statistical usage profile for each stimulus following every sequence of preceding stimuli. The Construction Plan Volume defines the sequence of increments in which the software is to be developed and certified.

8.2 The Specification Documents

There are two aspects to creating a specification.

The first is deciding what should be done and how to do it from the user's perspective. This is a hard problem. This is an inventive step. This where the look and feel of the product is determined. What functions will be offered. How the user will initiate each function. The list of inventions/decisions is enormous. In order to determine what should be done, there is the need to investigate the problem domain and the solution domain. Typically, these investigations require considerable time as alternatives are developed, evaluated, discarded and adapted until the final approach is selected. It is not possible to develop a prescriptive recipe to perform this inventive process. Typically, this difficult process is made even more difficult due to the fact that while all this exploration is proceeding, the project sponsors are modifying their view partially in response to the findings of the investigations and partially in response to other things and experiences they are having in the normal course of performing their jobs.

The second aspect of specifications is recording the results of all these inventions and decisions. These are recorded in the specification document. It is possible to be quite precise how the specification documents should be written. First the specifications should be formal, even if they are preliminary. Second, given document outlines to serve as an objective for recording the decisions/inventions made, it is quite easy to develop the specification documents.

Typical outlines for the six volumes are provided in Figure 8.2.1 (Figure 8.2.1 is 7+ pages in length). Each of the six documents has specific purposes, which define the content and format of the volume.

The purposes of the Mission Volume are (1) to define the mission that the software is to fulfill, (2) to define the context in which the software will be operating, and (3) to record the argument that says if the software satisfies the defined mission, it will accomplish its share of the goals assigned to the automation.

The purpose of the Functional (Black Box) Specification Volume is to define an implementation-free view of the internal structure of the software being designed. This implementation-free view is obtained by specifying functions which define the conditions in terms of stimuli histories that cause the presentation of each possible response that can be produced by the software.

The Functional Specification Verification Volume presents an argument which justifies the correctness of the Functional (Black Box) Specification Volume. This argument is not easy to make, but serves as the basis for accepting or rejecting the specifications.

This User's Reference Manual volume contains a precise definition of all the inventions made by the software designers in terms of the stimuli and responses invented for the software to interface with the environment in which it will be operating. A user with sufficient subject matter expertise should be able to use the software by using this volume.

The Expected Usage Profile volume contains the definition of how it is anticipated the software will be used by each class of user. This definition is required to develop test scenarios in accordance with usage specifications so it is possible to make measurements for how reliable the software will be when it is put into service in the environment where it is expected to operate.

The purpose of the Construction Plan volume is to develop a plan for the Development and Certification teams during actual product development. The plan contains the actual modules and functions to be a part of each increment. The result is a list of modules that the Development team needs to produce for each increment, and the functionality available for the Certification team to certify for each increment.

Figure 8.2.1 Software Specifications: Typical Outlines

Software Specification For

[Software Project Name]

Volume I - The Mission

Section 1 The Mission Statement

A mission statement identifies the purpose of the service that the software is to provide to the organization (product) using the software. The mission statement specifies what effects the software is intended to have on each of the stakeholders in the service the software is providing. A mission statement should rarely be more than one page. It should be a clear, concise statement of intent. A stakeholder is any noncompetitive individual or organization directly affected by the software's behavior and performance; for example, stockholders (it is a rare software system that does not affect the profit), customers, suppliers, distributors, employees, government agencies, etc.

A typical profile for a mission statement is:
The mission of the __(name)____ software system is to provide ______(the service)_______ to _____(the users)______ in such a way as to satisfy:
1. the shareholders by ____
2. the software users in department ____ by _____
etc. for all stakeholder.
The above structure tells exactly what service the software is intended to provide to whom and defines the direct and indirect benefits and costs to each individual and organization affected by the software. Such a statement should be challenging and exciting to virtually all of an organization's stakeholders.

Section 2 The Detailed Mission

This section defines the requirements allocated to the software so the software is in a position to satisfy the mission statement. This section should define each requirement for the software in a single statement in the format:

The software shall ____. Each requirement should be followed by

1. a justification for how the subject requirement contributes to fulfilling the mission assigned to the software,

2. a rationale for the choice of any quantitative limits containe.l in the statement so it is clear why that value has been selected rather than the quantitative target plus or minus a delta, and

3. a definition of how the accomplishment of the requirement can be observed in the service provided by the software.

Figure 8.2.1 Continued on Following Page

Figure 8.2.1 Continued

Section 3 The System Context

This section defines the context or the environment in which the software will be operating. The precise contents of this section will differ depending on the situation. The following is a list of topics which typically may be included:

1. a description of the current system to be improved by the planned automation,
2. how the new system that includes the new automation components will be operating,
3. the expected benefits, and
4. a summary of the other components of the system so it is clear how the software will fit in with other system components.

The exact contents will depend on what is required to insure that the contemplated role of the software in the environment to be created is clear.

Section 4 The Mission Validation Argument

This section contains the arguments that say if the software is implemented and accomplishes the defined mission, the software will accomplish its intended purpose in the environment where it will be operating.

Software Specification For

[Software Project Name]

Volume II - User's Reference Manual

Section 1 The Environment

The software exists in a larger environment. This section defines the environment that the software will operate in. This section typically includes subsections on:

- the computing hardware
- the peripherals
- the directly connected devices if any
- the local and remote communications network, if any
- the operating system environment
- related software
- related data bases
- the people using the software

Figure 8.2.1 Continued on Following Page

Figure 8.2.1 Continued

Section 2 The Stimuli

This section defines many important inventions that have been made to make the software useful to fulfil its intended mission. The exact outline of this section depends on the organization that will best explain the inputs to the software. But no matter what the organization is, this section must define the following for each stimulus:

The precise syntax and format of each stimulus.

The method by which the stimulus is developed from the environment. For people-generated stimuli, this transformation needs to deal with the anticipated working relationships of the people generating the stimuli. This typically means defining or referencing the manual systems and procedures to put the software stimuli into context. For analog stimuli, this means defining the precise way the device communicates with the software including all issues of timing.

Key points and issues the user needs to understand when using that stimuli.

It is often very desirable to include examples that help the reader understand the use of the stimuli. In addition to the above information which is normally best defined one stimuli at a time, a formal definition of all the stimuli using formal grammars that integrates the relationships between the stimuli must be included.

It is very important to organize this section for readability and understandability. Typically, it is difficult to make clear the relationship between all the invented stimuli so the organization is very important. To enhance readability it is often necessary to included references to other sections which define other stimuli and responses.

All stimuli including those for initiating use of and terminating use of the software must be defined. No stimuli should be left for latter invention. A common failing of many software designers is to first specify and then implement the mainline stimuli/responses and then add the special commands required to perform other activities. This policy leads to undesirable consequences.

Section 3 The Responses

The exact outline of this section depends on the organization that will best explain the outputs from the software. But no matter what the organization is this section must define the following for each response:

The precise syntax and format of each response. It is often very helpful to define reports, screens, menus, etc. in terms of formal grammars.

Figure 8.2.1 Continued on Following Page

Figure 8.2.1 Continued

The method by which the response is used in the environment. For people consumed responses this transformation needs to deal with the anticipated working relationships of the people using the response. This typically means defining the manual systems and procedures to put the software responses into context. For analog responses this means defining the precise way that the software communicates with the devise including all issues of timing.

Key points and issues that the user needs to understand when using that response.

It is often very desirable to include examples that help the reader understand the use of the response.

In addition to the above information which is normally best defined one response at a time a cross reference that defines which responses can be caused by each stimuli. This information is defined precisely by the Black Box function but this summary information is often very helpful in communicating understanding as well as being helpful in validating the external design of the software.

It is important to organize this section for readability and understandability. Typically it is very hard to make clear the relationship between all the invented stimuli so the organization is very important. To enhance readability it is often necessary to included references to other sections which define other stimuli and responses.

Section 4 System Performance

In many systems there are performance constraints associated with the transformation of stimuli into responses. These requirements are typically related to speed of transformation of the accuracy or precision of the transformation. All performance standards must be defined.

Section 5 Undesired Events

Software is designed to handle undesired events as well as the desired or planned events. In this section all undesired events are enumerated and the response that will be taken to each undesired event is defined along with the corrective actions the user may take in those cases where corrective actions are required.

Section 6 Software Initiation

This section describes what to do when the software arrives from the developer. It describes how the software is to be installed, configured or tailored, how files are to be initialized, and how the software is invoked for the first time, what to do in case of a failure and the system must be re-initialized and all other activities associated with making the software ready for use.

Figure 8.2.1 Continued on Following Page

Figure 8.2.1 Continued

Section 7 Glossary

What is meant by

Section 8 Index

Where to find it

Software Specification For

[Software Project Name]

Volume III - Functional (Black Box) Specification

Section 1 The Stimuli

This section lists all the stimuli and for each stimulus provides an exact name, a brief description, a reference name or number used in the Black Box function for identification purposes, and reference section(s) in Volume III of the specifications which define the stimuli in full detail.

Section 2 The Responses

This section lists all the responses and for each response provides an exact name, a brief description, a reference name or number used in the Black Box function for identification purposes and reference section(s) in Volume III of the specifications which define the responses in full detail.

Section 3 The Black Box Function

This section contains a sub-section for each stimuli. Each sub-section defines the conditions in terms of stimuli histories under which each possible response that can be triggered by the arrival of the stimulus which is the subject of this sub-section. The Black Box sub-functions can be expressed in terms of pseudo-code or program tables.

It is very important that these functions be defined very carefully and are well written so that everyone who is interested in the software can read them with comprehension. It is very important that no state data or other implementation-dependent inventions creep into the definition of the Black Box functions because if they do, they are no longer Black Box functions. Once they become something besides Black Box functions, they lose their independence and ease of comprehension.

Figure 8.2.1 Continued on Following Page

Figure 8.2.1 Continued

Software Specification For

[Software Project Name]

Volume IV - Functional Specification Verification

The Black Box Validation Argument

This document contains the argument that says if software is implemented so that it satisfies the defined Black Box functions the software will fulfill its intended mission.

It is important that this argument be well presented since it is possible to verify the software as developed to the Black Box function. To insure the software will satisfy its intended mission, it is necessary to satisfy all interested parties that if software satisfies the given Black Box function, it will satisfy its intended purpose.

Lower level design work should not progress until all interested parties are satisfied with the Black Box function. That does not say that bottom-up exploration can not proceed. It is often necessary to conduct extensive bottom-up exploration to develop the specification, but bottom-up exploration should not be confused with top-down design.

Software Specification For

[Software Project Name]

Volume V - Expected Usage Profile

Section 1 Usage States

As the software operates, it moves from one program state to a next program state. For example, in a simple situation the software is in the state of displaying screen 15 and as a result of some action moves to another state by displaying screen 22. The first step in developing a usage profile is to specify all the program states that the software can be in. This list of program states can be developed by an examination of the Black Box Function and Users Reference Manual. Program states have nothing to do with actual implementation. These are states defined by the understanding of the problem and solution domains of the system to be implemented.

Figure 8.2.1 Continued on Following Page

Figure 8.2.1 Continued

Section 2 Stimuli and Stimuli Distributions

In order to describe the transitions between the program states, it is necessary to understand the nature of the stimuli to the system to be implemented. The potential stimuli and their distributions for each state need to be listed. This needs to be preserved as it will be used in order to generate test scenarios.

Section 3 Usage Profile

The program execution paths can now be converted into the expected usage profile. The format of a usage profile is a matrix of transition probabilities. The matrix is an n x n matrix (where n program states have been identified) with each row and column representing a program state. The rows represent from program states and the columns to program states. The cells of the matrix represent the transition probabilities of software moving from program state i to program state j. The transition probabilities along each row must sum to 1.0. Typically only a small percentage of the cells have non-zero transition probabilities. Once this matrix is available, it is easy to generate test scenarios for the defined user class. If there are multiple classes of users, one transition probability matrix can be obtained by determining what proportion of the usage will come from each class of user.

Section 4 Transition Processes

For each possible transition, it is necessary to define the sequence of stimuli that cause the transition to occur. This information is required to generate test cases.

Section 5 Assumptions

In developing the usage profile, certain assumptions will have been made. These assumptions are recorded in this section.

Section 6 Increment Usage Profiles

The software will be constructed and certified in increments. Each increment is executable by user commands. In order to measure the reliability of each increment, it is necessary to develop a usage profile for each increment from the usage profile for the entire software system. The expected usage profiles for each increment are recorded in this section.

Figure 8.2.1 Continued on Following Page

Figure 8.2.1 Continued

Software Specification For

[Software Project Name]

Volume VI - Construction Plan

Increment 1:

1. Specifications references for increment 1

2. Increment Architecture

    a) First level black boxes
    b) Second level black boxes

    ...

3. Comרonent 1

    a) Black Box for Component 1
    b) Additional material available on Component 1

4. Component 2

    ....
    ....

....
....
....

Increment 2:

....
....
....
....
....
....
....

End of Figure 8.2.1

8.3 Recording The Specification

In this section no attempt is made to define how to go about recording the specification. It is a detailed writing job. Section 9 gives guidance in developing a Black Box function which is the subject of volume 3. It typically requires a great deal of hard thinking to develop the Black Box function. Section 8.4 provides guidance in developing a usage profile.

The verification volume may be a new idea to some software engineers. In many environments, the software design is given to the users, problem sponsors and others; and they are expected to appraise the design and determine if it will solve their problem. This mode of communication puts the responsibility for proving the specifications are acceptable on the user. In this manual, it is suggested that it is better to put the responsibility for arguing that the specifications define a product which solves the user's problem on the designers. This seems to be the proper allocation of responsibility since it is the designers who made the inventions that they believe will solve the specified problem.

It is important to note that the specification document must be done before the final design is started. If it is not done, there is no way to verify the software; and even more important, the software designers do not know in detail what they must do.

8.4 Developing the Expected Usage Profile

A Markov process is defined by an n by n matrix where the rows and columns represent the states in which any system can be in. The cells of the matrix represent the probability of moving from state i to state j. The probabilities on any one row must sum to 1.0, except for terminal states which are defined by a row of zeros since there is no probability of moving from a terminal state to an active state. Repeated multiplications of the matrix will finally result in a matrix with all non-zero elements on the diagonal. These values represent the steady state probability of the system being in state i. For example, the following simple model that appears in Figure 8.4.1, using a Markov process, might represent all possible realizations of a glider flight.

Figure 8.4.1 Program States for a Glider

Flight	Up	Down	Sideways	Crash
Up	.25	.50	.25	.00
Down	.40	.25	.25	.10
Sideways	.50	.50	.00	.00
Crash	.00	.00	.00	.00

The Markov process is a powerful abstraction (model) for representing all the possible realizations of a software system. Consider the following:

> With a statistical usage specification, the probability of each stimulus in each state is known.
>
> The functional specification defines what the state will become, as will the response to the user.
>
> These two facts define a Markov process, given
> the set of all states, and
> probability of going from one state to the next.
>
> The interaction of a user with a system defines a particular realization of the Markov process.
>
> Each selection of a stimulus (command and/or data) defines a Markov transition.
>
> The probability of the entire sequence will be the product of the probabilities of each transition.
>
> Responses play no role in the Markov process except to provide information for the user selection of the next sequence of stimuli.
>
> Entire lifetime realizations of a software system will be a sequence of states defined by a Markov transition.
>
> The same software can be used to obtain many realizations, each a unique history of stimuli which have created the states.
>
> These realizations will be different and failure experiences, if any, will be different as well.

From the reasoning above, it is clear that a Markov process can be used to model all the realizations of a particular software system. A system of programs where the realizations of the program can be modeled by a Markov process has the property that when the system is in any program state, the next program state can be predicted without regard to previous states. It is possible to find program states that represent all possible realizations for any software system. Therefore, a Markov model for a software system can serve as the basis for usage testing for that software system.

Then to prepare usage tests, only two issues pertaining to the software need to be understood:

1. Program States - the status of the program at any one time.
2. Stimuli - what is being received next by the program to cause it to move to the next program state.

The first task in developing a Markov model is to define the program states. A program state is the status of a program from the user's perspective. The program state can always be determined from the Black Box (specifications) for the software system. One does not need to know anything about the actual structure of the program to be executed. A tester would like to characterize the status of the program and the next status in which the program can exist. The goal is to calculate the probabilities of going from one program state to the next (transition probabilities).

Characterizing all of the program states is not trivial. A system can have a large number of states. In these cases, program states can typically be represented by equivalence classes, each of which represent multiple states. For example, if we are considering the sum from throwing two dice, the states (1,6), (2, 5), (3, 4), (4, 3), (5, 2) and (6, 1) can all be viewed as program state 7.

A user moves from program state to program state by stimuli. To understand stimuli, one must understand the probability of what the next user stimulus will be, given any program state. These probabilities will add up to one. Each stimulus (or set of stimuli) will take you from one program state to another (maybe the same state). The probability of moving from program state a to program state b is the sum of the probabilities of the stimuli that will move you from program state a to program state b being selected.

The program state analysis does a variety of things:

a) It randomizes control flow, and can also randomize data values.
b) Equivalence classes for data values can also be determined.
c) This includes looking at random spellings of a possible input word, and looking at classes of valid, invalid and non-unique names.

At this time, a blank n by n matrix has been defined, which represents all program states. At this time, one needs to determine the transition probabilities. This can be done in a variety of ways. Observing another similar system in actual field use often gives a good assessment of what the usage profile is like. Other methods of determining operational usage include interviewing or polling individuals familiar with the intended use of the software system and developing and using a prototype. In the case where the system to be developed is an enhancement or an extension of an existing system, the usage profile of the existing system can be used as a starting point to be modified in an incremental manner through the use of the new features. This type of approach will generally give an extremely accurate reflection of the expected operational usage of the software system to be built. A matrix of transition probabilities is shown in Figure 8.4.2. Very often it is more desirable to show only the non-zero transition probabilities in a table form as shown in Figure 8.4.3.

If one of the above options is not available, usage profile can be estimated by analyzing execution paths.

Execution paths are sequences of stimuli which cause state transitions. The state transitions may leave the user in the same state. The algorithm for constructing Markov transition probabilities from the execution paths is as follows:

for all states

1. Determine execution paths/trees by starting at a state, and listing unique stimulus (i.e., keystrokes, sensor inputs) combinations that will result in a state transition. Some stimulus combinations may be a part of an equivalence class.

2. Count the frequency of each state/stimulus used, or associate a probability with the selection of each stimulus.

3. Take the sum of all stimuli that take the program from one state to another. Enter that sum into the corresponding position in the transition matrix. Divide each value in a row by the sum of the row (to normalize the frequencies). Rows in the matrix (i.e., the probability of a state transition) must add up to one, except for terminal states.

Figure 8.4.2 State Transition Matrix

		State To:											
		1	2	3	4	5	6	7	8	9	10	11	12
S	State 1	.25	.25			.25			.25				
t	State 2	.25			.25	.25		.125				.125	
a	State 3		.50						.50				
t	State 4					.50			.50				
e	State 5	.25	.25	.25			.125			.125			
	State 6		.25						.25		.50		
F	State 7					.50						.50	
r	State 8	.25		.125	.125			.125		.125	.25		
o	State 9		.50				.50						
m	State 10												1.0
	State 11					.25			.25		.50		

Figure 8.4.3 Typical Usage Profile

From State	To State	Transition Probability
S1_INITIAL	S1_UP_ARROW	0.17
S1_INITIAL	S1_DOWN_ARROW	0.66
S1_INITIAL	S1_ENTER_PROJECT_MANAGER	0.17
S1_UP_ARROW	S1_ENTER_EXIT	1.0
S1_DOWN_ARROW	S1_DOWN_ARROW	0.44
S1_DOWN_ARROW	S1_ENTER_NETWORK_DEFINITION	0.12
S1_DOWN_ARROW	S1_ENTER_RELIABILITY_INFO	0.11
S1_DOWN_ARROW	S1_ENTER_CERTIFICATION_MODEL	0.11
S1_DOWN_ARROW	S1_ENTER_EXIT	0.22
S1_ENTER_PROJECT_MANAGER	S2_INITIAL	1.0
S1_ENTER_NETWORK_DEFINITION	S3_INITIAL	1.0
S1_ENTER_RELIABILITY_INFO	S4_INITIAL	1.0
S1_ENTER_CERTIFICATION_MODEL	S5_INITIAL	1.0
S1_ENTER_EXIT	S1_INITIAL	1.0
S2_INITIAL	S2_UP_ARROW	0.14
S2_INITIAL	S2_DOWN_ARROW	0.14
S2_INITIAL	S2_F1	0.14
S2_INITIAL	S2_F5	0.58

This three-step algorithm is explained in more detail in the following three paragraphs.

In constructing execution paths, one must determine what is the usage of the system. Options include: paths that a 'deliberate user' would follow, random paths, and paths based on the probability that a stimulus would be selected. The execution paths can be represented in a textual (called execution paths) or a graphical fashion (called decision trees). Examples of both representations appear in Figure 8.4.4.

Once execution paths have been determined, then state frequencies are calculated. State frequencies denote the raw number of uses of a particular stimulus in a usage path. Dividing the frequency of a particular stimulus by the sum of the frequencies for all stimuli that can appear in a certain program state gives the probability that the particular stimulus will occur.

As a final step, the state frequencies are entered into a matrix. Sums are then taken for each row and then each element in the row of the matrix is divided by the sum. The result is now the Markov Transition matrix. Most of the values in the matrix will be zero although all rows, except terminal states, add up to one. This step is illustrated in Figure 8.4.5.

When the Markov model is complete, the long run probability distribution for the states can be calculated by generating the Markov chain for the matrix. The long run probabilities will list the states that are not used much, which may justify the elimination of parts of the system. Additionally, the long run probabilities show, on the average, how many test cases must be run to function certain features.

The process of creating a usage profile may initially seem to be an intimidating or impossible task. In fact, the process is just a modeling problem. It may not always be a simple modeling problem, but it is just a modeling problem. The steps in the process that are mentioned define an algorithm for determining the usage profile. Usage states and transition probabilities will be determined through analysis. A number of relatively complex software systems have been described in this manner, the task of creating that description was not always trivial but was always possible to do in a manageable amount of time.

Figure 8.4.4 Execution Path and Decision Tree Examples

```
Initial-1
  Project Manager ; Screen-2
  Up-arrow ; Exit System
  Down-arrow
     Network Definition ; Screen-3
     Down-arrow
        Reliability Information ;    Screen-4
        Down-arrow
           Certification Model ; Screen-5
           Down-arrow ; Exit System
```

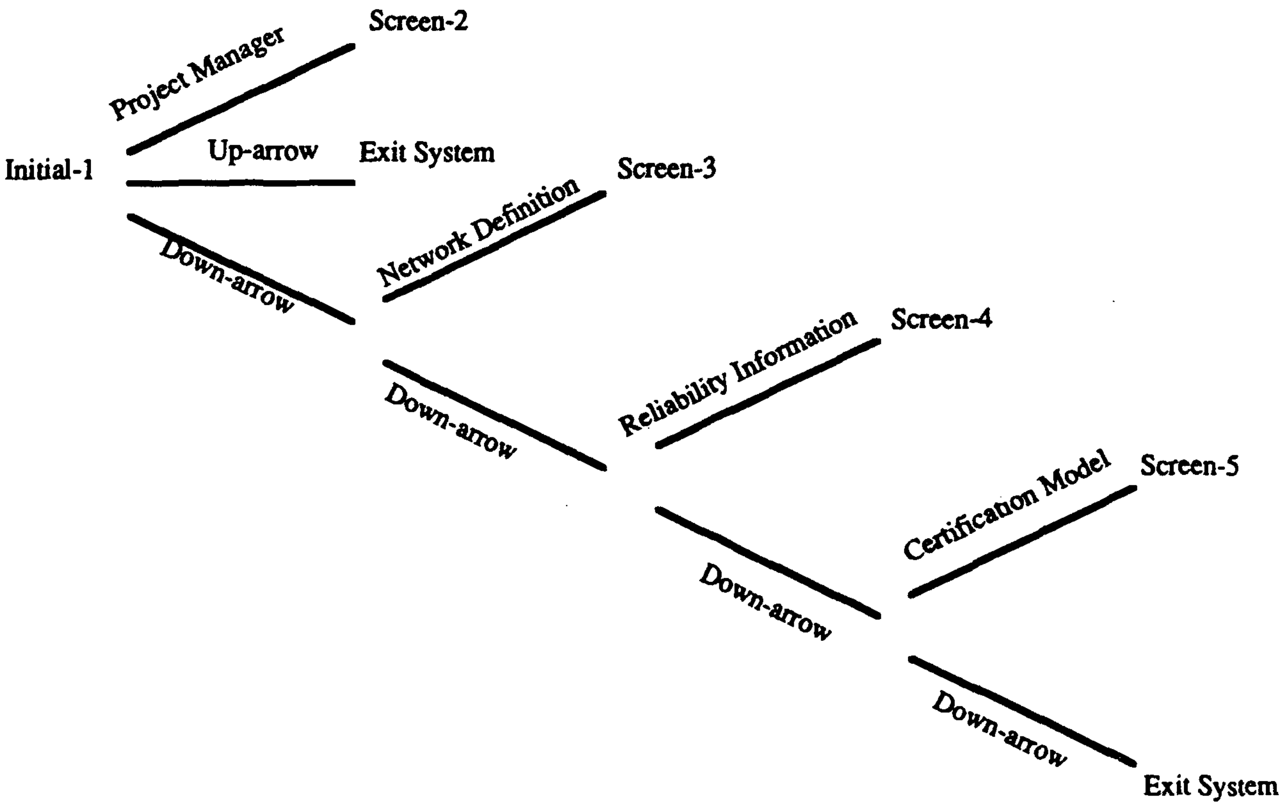


Figure 8.4.5 State Frequencies and State Probabilities

From	To					Sum
Initial-1	PM	Up-arrow	Down-arrow			
	1	1	4			6
	1/6	1/6	4/6			1
Down-arrow	ND	RI	CM	Exit	Down-arrow	
	1	1	1	2	4	9
	1/9	1/9	1/9	2/9	4/9	1

Section 9
Cleanroom Development Team Practices

In this section the Cleanroom Project cycle is examined from the vantage point of the Development team. In the Cleanroom environment the development team has a well-defined mission which is stated as follows:

> Given a function (specification) which is to be implemented in software, find a rule (the program) that correctly implements the function.

In performing this mission the Development team is performing the engineering tasks defined in Process E18: Develop Increment i and in Process E24: Correct Code Increment 1...j in section 4. The Development team also contributes to performing the work defined by the three processes that have to do with understanding the problem and solution domains:

E12: Understand Problem Domain
E13: Understand Solution Domain
E21: Increase Understanding of Problem and Solution Domains as Required

The principal effort is devoted to accomplish Process E18. This section is devoted to discussing their performance of the engineering tasks associated with this process. When the Development team contributes to the other processes in the capacity of software developers, they use the same software development capabilities as discussed here. When they are performing other tasks on the software project, they may use any of the tools available to software engineers or to other engineers and analysts. These are not discussed in this section. In addition, the Development team contributes to performing the project overhead tasks as required. These include

E7: Maintain Project Schedule
E8: Prepare for and Conduct Project Review
E9: Prepare and Submit Status Reports
E11: Submit a Question or Issue.

The Development team or teams implement the specification for the entire software system increment by increment. The construction plan, which is volume six of the specifications, has decomposed the specifications into sub-specifications, one for each set of accumulating increments. That is, increment 1, increments 1 + 2, increments 1 + 2 + 3 and so on. A Development team is responsible for each increment. In the balance of this section, discussion is limited to the Development team working on a single increment without any loss of generality since an increment is the basic work unit for a development team.

The Development team is provided with a well-defined specification for the increment. For a specification to be well defined, it must clearly define the function that the software is to perform. The Development team must then find a rule that efficiently implements the specification. In addition, they must verify by logical/mathematical argument that the rule found correctly implements the function. They perform verification by logical argument to minimize to the maximum extent possible the most error-prone activity associated with software development - debugging.

The Cleanroom environment provides the Development team engineers with several important features facilitating their search for a rule that correctly implements the specification. These facilities include:

Developers use an iterative approach in software development by performing many small steps on the path of finding a correct rule for the given function. Each small design step is then immediately followed by a verification to show that the function resulting from the design step is equivalent to the function prior to the design step. This process is called Stepwise Refinement With Verification. The expansions and verifications at each step are guided by mathematical principles that computer programs and systems of computer programs must observe.

Developers use well-developed mathematics to guide software development and verification. The mathematics of a computer program is developed in Structured Programming: Theory and Practice by Linger, Mills and Witt, Addison-Wesley, 1979. The mathematics of a system of computer programs is developed in Principles of Information Systems Analysis and Design by Mills, Linger and Hevner, Academic Press, 1986. An algorithmic procedure to design software systems which exploits the mathematics of a system of programs as developed in Mills, Linger and Hevner is defined in "Stepwise Refinement and Verification in Box-Structured Systems" by Mills, IEEE Computer, June 1988. Specifically, the Development team uses Box Structures to allocate the function for the entire increment to the programs that make up the increment and determine what state data to invent to act as surrogate for the stimuli histories. Then once the functions for each Clear Box have been determined the functions are further refined until all subfunctions connected by programming statements are executable in the target programming language. Both of these strategies revolve around using Stepwise Refinement with verifications between each such refinement.

Developers work in a team since it has been found that verification arguments or proof conversations are best performed in a group rather than on a solo basis.

Developers do no compilation or computer testing in order to avoid the most error-prone activity associated with software development. It has been observed that when a change is made to a software program to correct a malfunction in a test or debugging environment that often the result is a correction of the currently observed problem and in addition a deeper logic error is inserted in the software that will cause a failure in subsequent usage. The two events of making the fix and observing the failure induced by the fix are often separated by a great deal of time. A rule of thumb used by some software development organizations is that on the average one time in five the result of a software fix is that a deeper logic error is unintentionally designed into the software that will be observed sometime latter. These organizations believe that certainly as often as one time in ten the result of the fix will be deeper logic error that will cause a subsequent failure. It is these odds that make advanced software development organizations attempt to minimize the number of fixes made to software. This relationship between fixing observed failures and introducing new faults that will result in failures to be observed subsequently that make it nearly impossible and certainly very expensive to test quality into software. Additionally, since developers do no testing it has been observed that they are more careful in the design and coding process. That is why in the Cleanroom approach the stress is on designing in quality and minimizing corrections due to testing.

The general concepts defined above give the Development team the ability to write largely failure-free software. In subsequent subsections, those topics will be discussed in detail in the order listed below:

1. Stepwise Refinement
2. The Box Structures Design Algorithm
3. Box Structures Verifications
4. Stepwise Refinement Using Structured Programming
5. Structured Programming Language Verifications
6. Conducting Proof Reviews
7. Working With The Certification Team
8. Working With The Specification Team

9.1 Stepwise Refinement

The Software Development Problem: Given a function, say F_0, which is to be implemented in software in a given programming language for a given computer.

The Stepwise Refinement Algorithm:

Initialization: Let i = 0.

Step 1: Find a new function F_{i+1} which is equivalent to F_i where some part of the F_i function has been decomposed into subfunctions connected by programming statements.

Step 2: Show that function F_{i+1} is equivalent to F_i.

Step 3: If all the subfunctions in the new function F_{i+1} are executable in the desired programming language, terminate with the desired rule for the function; otherwise, increment i and loop to Step 1 to continue the stepwise refinement algorithm.

Verification Of Program Correctness:

Assume the algorithm terminates after n iterations with function F_{n+1}, which has been shown to be equivalent to F_n, which has been shown to be equivalent to F_{n-1}, and so on to F_1, which has been shown to be equivalent to F_0 which is the given function. By this chain of logical reasoning, if no human error occurred, the program is a correct rule for the given function.

As we all know, humans are fallible so it is possible that the rule as found does not correctly implement the function. The question arises that if humans are very error prone, the Stepwise Refinement Algorithm would not be very useful since most times the resulting rule would be an incorrect rule for the given function. The pleasant surprise is that when the software developers work as a Development team using this simple algorithm guided by the proof rules as developed in Structured Programming: Theory and Practice, they have a high probability of finding a rule that correctly implements the function in the vast majority of the situations.

9.2 The Box Structures Design Algorithm

The software design problem can be described in four questions:

a) What objects?
b) What secrets to hide in each object?
  - What data to encapsulate?
  - What processes to supply?
  - What algorithm to use for each purpose?
c) What is the consumer/supplier relationship between objects?
d) What is the structure of the interface between objects?

Four necessary conditions are currently known which a solution to the software design problem must observe:

a) Hide all data in data abstractions or objects.
b) Define all processing by sequential or concurrent usage of data abstractions.
c) Migrate usage of each data abstraction to the lowest place possible in the usage hierarchy.
d) Maintain referential transparency between defining responses in terms of stimuli histories and the state data selected to be used to represent the stimuli histories.

As a result of understanding the design problem and the necessary conditions, a strategy for box structure design becomes clear. One must directly solve the software design problem subject to identified necessary conditions. Also, one must use stepwise refinement and verification to establish intellectual control over the inventive process.

The Box Structure design strategy is to build the module structure top-down inventing state data, state data maintenance functions, process performance functions and intermodule communications in the logical top-down stepwise expansion, ordering with verification at each step.

A Black Box description of a data abstraction is a function on the set of stimuli. Black Box functions are implementation free. The function to be performed is stated entirely in terms of the inputs (stimuli) that the abstraction has received. The Black Box defines all responses in terms of stimuli histories for this and all other stimuli. For that reason, each subfunction is defined in the notation of S*||Si, which defines the sequence of all previous stimuli along with the present stimulus Si. A Black Box sub-function to change the password for a buoy system (from IEEE Computer, June 1988) appears below:

Black Box sub-function for S*||S4: change password command and data is

```
E4:  if  no shutdown command since last restart command
         then
             if  password correct * and no restart command or shutdown command
                 then
                     confirm password change
             fi
     fi
```

* where password correct represents the password most recently entered during a password change, or is 'buoy' if a password change has not been a part of stimulus history

Black Boxes are a completely general means of describing and analyzing behavior. Their focus on expressing behavior in terms of stimuli and responses describes how users actually deal with a system -- free of all internal processing details. Interactions between Black Boxes are completely arms length -- no shared data -- a supplier/consumer relationship.

With a State Box, data abstractions store data between stimuli in order to respond to the effect of prior stimuli. Information hiding requires that such stored data be part of the abstract state of the abstraction. Data can be stored in any form as long as it correctly represents all previous stimuli. One such correct representation is to store the history of all prior stimuli.

Classical state machines are not useful in hierarchical systems when data needs to be migrated to different levels in the hierarchy. The State Box generalization provides the solution by determining the state and response for each stimulus. State Boxes can then be distributed throughout the usage hierarchy. State Boxes begin to add implementation details. State data to act as a surrogate for stimuli histories have been selected to be maintained at this level. The Black Box function is rewritten by substituting state data as appropriate and to reflect required state data maintenance. An example of a State Box sub-function appears below (repeated lines from Black Box are shown in bold text):

State Box sub-function for S4: change password command and data is

```
E4:  if  'buoy_status' on
         then
             if  password in stimulus = 'password' and no restart command or
                 shutdown command
                 then
                     con
                         confirm password change
                         update 'password'
                     noc
             fi
     fi
```

In this notation, 'buoy_status' and 'password' had been selected to be state data that will be stored at this level, and are serving as surrogates for a part of the stimuli histories that pertain to whether the buoy is on or off and the current password in storage.

The next step in defining the data abstraction is to define the State Box transformation function in more detail -- in terms of the Black Boxes to use at the next lower level in the hierarchy. Theorems and experience with structured programming lead to direct definition of four kinds of Clear Boxes: sequence, alternation, iteration and concurrency. All of the theory and experience with structured programming and stepwise refinement carry forward into Clear Boxes. Clear Boxes can be documented in many forms, including: English, pseudo code, flow diagrams, and a programming language. Clear Box functions complete design details for that level. To derive the Clear Box, the State Box function is modified to (1) complete details for maintenance of state data and (2) invocations of lower level Black Boxes. An example of a Clear Box (from the same buoy problem example) appears below:

```
Clear Box buoy
        .
        .
        .
        case stimulus is
            .
            .
            .
            part S6: air temperature data
                con
                    acknowledge air temperature data
                    use air_temperature(S6, 'air temperature running average') to update the air temperature
                    data
                noc
            .
            .
            .
end Clear Box buoy
```

A notation has been devised for Black, State and Clear Boxes. This notation can appear in two formats: as pseudo code and as a program table. The pseudo code approach is the style seen in the examples above. The formal notation for Black, State and Clear Box pseudo code appears in Figure 9.2.1.

A program table describes the logic in terms of conditions and data value modifications. The table lists possible conditions on the x-axis and responses and state data/local variable on the y-axis. For each condition, the corresponding data value changes are noted. Two alternate notations for a Black Box program table and a State Box program table appear in Figure 9.2.2. Program tables are also useful designing resolve logic.

Some organizations may want to modify the notation just presented in order to make it fit better in the organization's environment. The notation does not need to be identical to the format presented earlier, but it does need to be just as formal, readable and rigorously usable.

The Box Structure design algorithm is defined in Figure 9.2.3. At any step during the algorithm, it may be determined that any of the previous inventions were incorrect. In that case, that step will be redone in order to create a better invention. The process of design and redesign for one iteration of the loop will be done until the developer and Development team is convinced that they have created a correct set of inventions.

Once Clear Boxes have been created, they are refined, using the concepts to be defined in Sections 9.4 and 9.5. The refinements continue until correct and efficient code is developed. Each refinement needs to be verified to the previous refinement. Of course, the initial refinement is verified to the Clear Box.

Figure 9.2.1 Syntax for Box Functions

```
- - - - - - - - - - - ·BLACK BOX FUNCTIONS- - - - - - - - - - - -

black box <boxname>
stimuli:
  S1: <name> | <description/information> [: <type>]
...
responses:
  R1: <name> | <description>
...
begin black box function for S* || S: <boxname>


black box sub-function for S* || S1 : <stimuliname> is

  <black box pseudocode> | <black box program tables>

[end black box sub-function for S* || S1: <stimuliname>]
...
[end black box function for S* || S <boxname>]
[end black box <boxname>]


<black box pseudocode> ::= <black box statement> +

<black box statement> ::=


    <assignment statement assigning a constant to a response> |

    <definition of a response in terms of stimuli histories and/or previous responses> |

    if <condition in terms of stimuli histories and/or previous responses>
      then
        <black box statement> +
      [else
        <black box statement> + ]
    fi
```

Notes:
Reserved words: lower case in bold, only one per line
Indentations: two spaces
Stimuli, responses and black box functions are almost always recorded separately.

Figure 9.2.1 Continued on Following Page

Figure 9.2.1 Continued

```
- - - - - - - - - - - ·STATE BOX FUNCTIONS - - - - - - - - - - - - - - -
state box <boxname>
stimuli:
    S1: <name> | [<description/information>] | [: <type>]
...
responses:
    R1: <name> | <description>
...
state data:
    E1: <name> | [<description information> ] | [:<type>]
...
begin state box function for  S: <boxname>



state box sub-function for S1 : <stimuliname> is

    <state box pseudocode> | <state box program tables>

[end state box sub-function for S1: <stimuliname>]
...
[end state box function for  S <boxname>]
[end state box <boxname>]


<state box pseudocode> ::= <state box statement> <state box statement> *
<state box statement> ::=

        <assignment statement assigning an expression in terms of  constant(s), current stimuli, state
        data to a response> |
        <definition of a response in terms of state data, current stimuli, stimuli histories and/or previous
        responses> |
        <assignment statement assigning an expression in terms of constant(s), state data, current stimuli
        to a state data> |
        <definition of a state data in terms of state data, current stimuli, stimuli histories and/or previous
        responses> |
        if <condition in terms of state data, current stimuli, stimuli histories and/or previous responses>
           then
               <state box statement> +
           [else
               <state box statement> + ]
        fi |
        <concurrent control structure>
```

Figure 9.2.1 Continued on Following Page

Figure 9.2.1 Continued

CLEAR BOX FUNCTIONS

```
clear box <boxname>
     stimuli
          <stimulus name> : <type>
     responses
          <response name> : <type>
     state data
          <state data name> : <type>
     procedure <boxname>
          data
                 <local data name> : <type>
          begin
                 <procedure statements>
          end
     end procedure <boxname>
end clear box <boxname>
```

<procedure statement> ::= <assignment> | <control structure> | <box statement>

<assignment> ::= <variable> := <expression>

<control structure> ::= <sequence> | <alternation> | <iteration> | <concurrent>

<box statement> ::= use <boxname> (<stimuli name> <,stimuli name> * ,
<response name> <,response name> *)

Notes:
Reserved words: lower case in bold, only one per line
Indentations: two spaces.
Keywords: only one per line
Any consistent syntax for assignment, sequence, iteration, alternation and box statements are satifactory

NOTATION

|	or
::=	is an instance of
[ ]	optional
+	one or more occurences
*	zero or more occurences

End of Figure 9.2.1

Figure 9.2.2 Program Tables for Box Structures

black box sub-function for S*|| Si: <stimuliname>	conditions in terms of stimuli histories and/or prior responses	a separate column for each possible combination of conditions
first possible <responsename>	<constant> or <definition of a response in terms of stimuli histories and/or previous responses>	
a separate row for each possible response from this stimuli		

OR

black box sub-function for S*|| S1: <stimuliname>	conditions in terms of stimuli histories and/or prior responses	a separate column for each possible combination of conditions
responses	<assignment statement assigning a constant to a response> and/or <definition of a response in terms of stimuli histories and/or previous responses>	

state box sub-function for S*|| Si: <stimuliname>	conditions in terms of current stimuli, state data, stimuli histories and/or prior responses	a separate column for each possible combination of conditions
a separate row for each possible response from this stimuli	<expression in terms of constant(s), state data, current stimuli> | <definition of a response in terms of state data stimuli histories and/or previous responses>	
a separate row for each state data maintained at this level	<expression in terms of constant(s), current stimuli, state data> | <definition of state data in terms of state data, current stimuli, stimuli histories and/or previous responses>	

Figure 9.2.3 The Box Structures Software Design Algorithm

Define Black Box

1. Define stimuli (List all inputs to the system).
2. Define responses in terms of stimuli histories (Create the Black Box sub-functions and the software system responses).
3. Validate Black Box (For the first level Black Box, this is done by showing that problem description and Black Box function are consistent. The verification for the first level Black Box for the entire software system is recorded in the specifications. For lower level Black Boxes, this is done by showing that the Black Box function definition in the previous level's Clear Box and the Black Box function are consistent.).

Define State Box

4. Define state data to represent stimuli histories (Define surrogates that will preclude the need to continually handle stimulus histories by replacing the portion of stimulus history that needs to be preserved.).
5. Select state data to be maintained at this level (Determine which surrogates will be best stored at this level. This is a major step, and is a prime opportunity for the use of trade studies.).
6. Modify Black Box function to represent responses in terms of stimuli and the state data being maintained at this level (Create the State Box sub-functions, with Jata defined in terms of stimuli, state data and stimuli histories.).
7. Record type of references to state data by stimuli (Determine where and how each state data item is used by the sub-functions. This activity supports steps 8 and 9.).
8. Verify State Box (This is done by showing that the Black Box and State Box define the same function -- see section 9.3.).

Define Clear Box

9. Define data abstraction for each state data (Determine state data type, making it more concrete. The data types for the state data can include stack, integer, queue, array, etc.).
10. Modify State Box function to represent responses in terms of stimuli, this level's state data and invocations of lower level Black Boxes (Create Clear Box functions using stimuli, state data, local data, and lower level boxes).
11. Verify the Clear Box (This is done by showing that the State Box and the Clear Box define the same function -- see section 9.3.).

Loop

12. Continue until all Black Boxes are expanded.

Use of Box Structures provides:

a) Definition of objects and data abstractions.
b) Data abstractions migrated to lowest feasible level.
c) Data and process treated equally in each step.
d) Processing defined precisely for all objects.
e) Verification at each step of the design.
f) Real time processing.
g) Inventions separated into clearly identified small steps.
h) Details included throughout the design process so their late addition does not damage the design.
i) Complete design trail stored in Box Structure representation.
j) Intellectual control over the design process.

9.3 Box Structures Verifications

Mappings exist to map State Boxes into Black Boxes and Clear Boxes into State Boxes so each design step can be mapped back to its parent box to verify that the design is proper.· The verification procedure permits the designer to know the design is correct each step of the way. For stepwise verification, structured design using box structures must proceed from the top down. To facilitate thinking when designing and programming, humans need to explore ahead of the top-down design with bottom-up thinking.

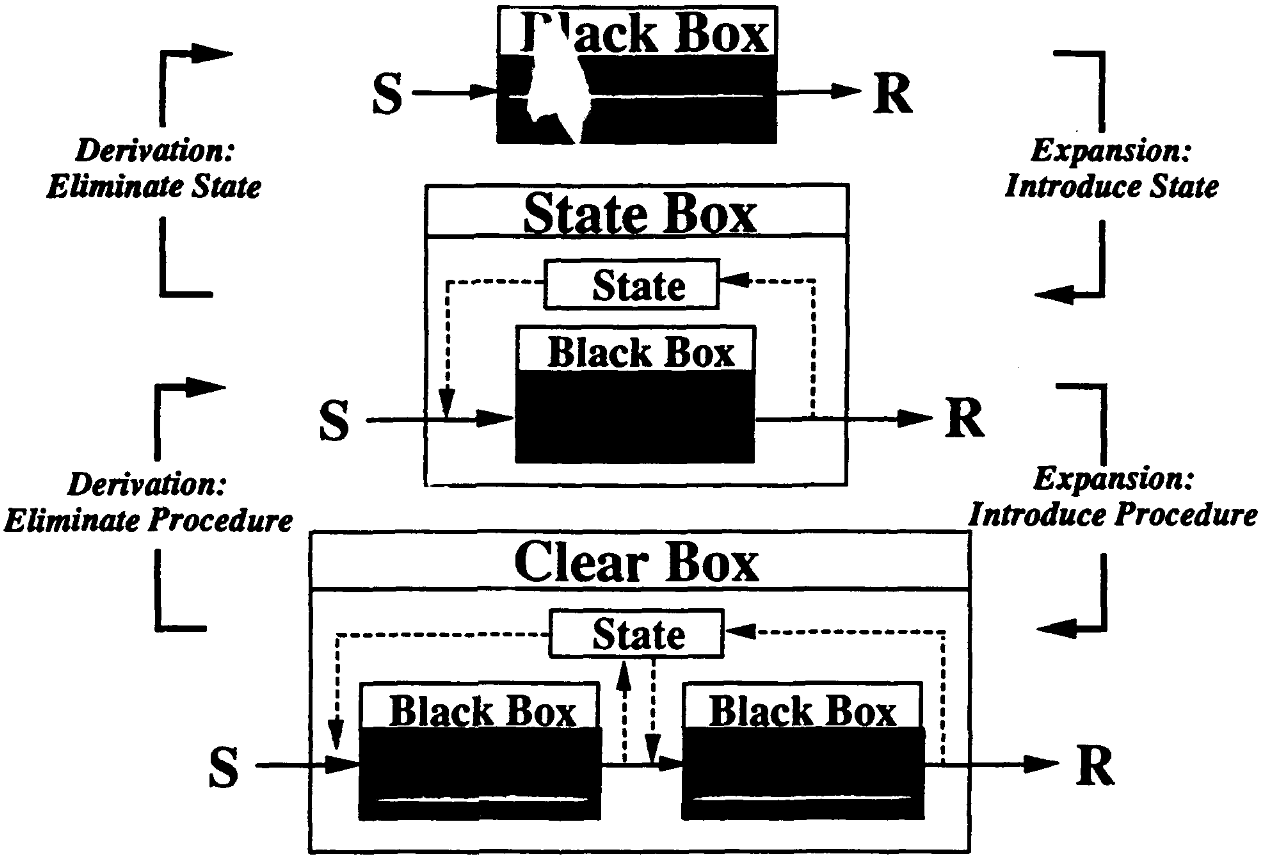
The strategy when verifying box structures is to re-derive the higher level box, and then compare the original higher level box with the re-derived higher level box. This concept is illustrated in Figure 9.3.1. The re-derivation process is made easier by using the following numbering scheme:

a) Number all Black Box statements B1...Bn.
b) For all State Box statements that did not appear in the Black Box, number those statements S1...Sn.
c) For all Black Box statements that were modified in creating the State Box, number those statements BSi, BSj, ...
d) For all Clear Box statements that did not appear in the black or State Box, number those statements C1...Cn.
e) For all Black Box statements that were modified in the State Box and are now modified in creating the Clear Box, number those statements BSCi, BSCj, ...
f) For all State Box statements that were modified in creating the Clear Box, number those statements SCi, SCj, ...
g) For all Black Box statements that were modified in creating the Clear Box, number those statements BCi, BCj, ...

To verify the state or Clear Box, one must check that the following the conditions are true:

a) Are all statements in the re-derived box that were modified from the original box (i.e., the BS, BSC, SC or BC statements) equivalent to those that were in the original box?
b) Are all lower level box statement that are new in the lower level box (i.e., the S statements in the State Box or the C statements in the Clear Box) all either keywords or used in confirming that condition a is true?

Figure 9.3.1 Expansion and Derivation Between Box Structures


Black Box
S
R
Derivation: Eliminate State
Expansion: Introduce State
State Box
State
Black Box
S
R
Derivation: Eliminate Procedure
Expansion: Introduce Procedure
Clear Box
State
Black Box
Black Box
S
R

c) Are the logical conditions for every higher level box statement (i.e., the B statements in the State Box and Clear Box, the S statements in the State Box, or the BS statements in the Clear Box) occur under the same logical conditions that they did in the higher level box?

9.4 Stepwise Refinement Using Structured Programming

The goal, once the Clear Box is derived, is to make a series of refinements that will lead to code, specifically, a structured program. Structured programs are programs which:

a) Have only one entry and exit
b) Use only sequence, selection and iteration statements.

Any flow chartable program can be expressed as a structured program. Structured programs can be wntten in any order -- it is only the order at execution that is important. Structured programs have mathematical properties that arise from the analysis of functions that permit people to reason about their correctness. The correctness or any construct in a structured program can be determined by reference to only the constructs that precede that construct in execution in the final program.

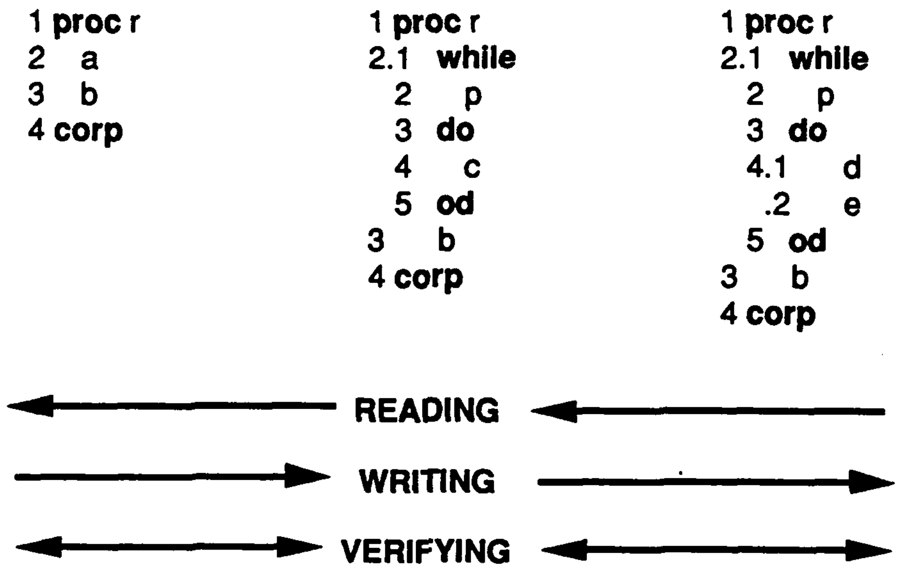
Experience indicates that humans typically develop more correct and complete programs when statements are developed for the final program 'top-down.' However, experience indicates that humans typically need to engage in 'bottom-up' thinking to refine their ideas prior to starting the final 'top-down' development of the program. The 'top-down' development approach is the basis of stepwise refinement. The approach is illustrated in Figure 9.4.1. When working with structured programs there are three operations that can be performed: function abstraction or reading, function expansion or designing and function comparison or verifying. When reading a program the reader is given a rule and then reads the program by deriving the function. When writing or designing a program the writer is given a function which is expanded into a rule. When verifying a program the verifier is given a function and a rule which are compared by deriving a function from the given rule and then comparing the derived function to the given rule. These ideas are diagramed in Figure 9.4.2

Figure 9.4.1 Stepwise Refinement

```
ORDER OF INCLUSION -- LEFT TO RIGHT
ORDER OF EXECUTION -- TOP DOWN

                        First Refinement        Second Refinement

proc [        ]     proc [          ]       proc [              ]

$F_0$                   if [        ]           if [            ]

endproc                     $P_1$                   $P_1$
                        then                    then
                            $F_1$                   $F_4$
                                                    $F_5$
                        else                    else
                            $F_2$                   $F_6$
                                                    $F_7$
                    fi                      fi
                    $F_3$                   $F_3$


                endproc                 endproc
```

Notes:
[ ] functional commentary

Figure 9.4.2 Program Abstraction, Expansion and Comparison



```
1 proc r
2  a
3  b
4 corp
```

```
1 proc r
2.1  while
 2     p
 3   do
 4     c
 5   od
3    b
4 corp
```

```
1 proc r
2.1  while
 2     p
 3   do
 4.1     d
  .2     e
 5   od
3    b
4 corp
```

The creation of structured code is done by selecting one of the structured programming constructs (sequence, iteration or alternation). Selecting the construct for a correct 'top-down' expansion must be well thought out. An example that presents a correct and incorrect expansion of a function into an initialized while loop is shown in Figure 9.4.3.

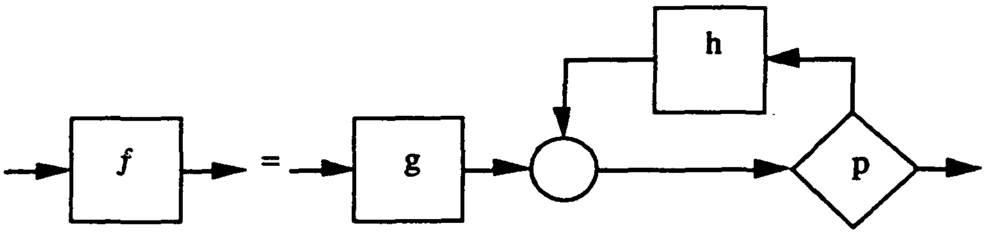
There are a number of rules that developers need to follow when designing and coding in order to increase the probability of writing correct code. These rules are often organization specific, although a number of them are probably universal. Useful rules are listed in Figure 9.4.4.

Code must be commented. Functional commentary typically takes a variety of forms. With Cleanroom, the functional commentary is the previous level of refinement, since that defines the intended function that the code (or lower level of design) will implement. Additional commentary can appear after structures in order to represent the post-condition of an implemented function. The functional commentary needs to be formal and precise so that the developer will be able to verify that the implemented function is equivalent to the intended function (that is, the functional commentary).

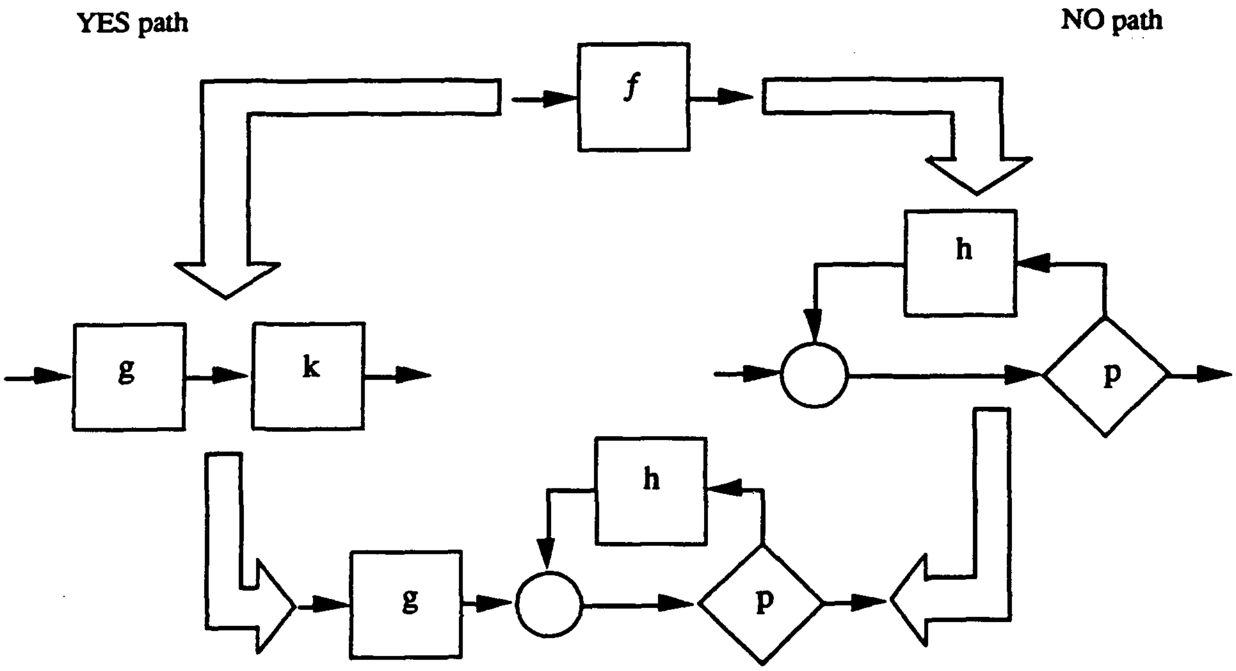
A number of benefits occur when using the above-mentioned concepts of structured programming. Programs are easier to write, easier to read, easier to verify and cost less. Also, software development moves from a private art to a public science. Finally, design and code written in this manner begins to give a firm basis for easier maintenance and reuse.

Figure 9.4.3 Stepwise Expansion

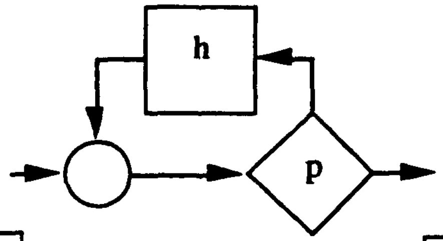
- Given function *f* to expand, with foreknowledge it will end up with an initilized iteration:



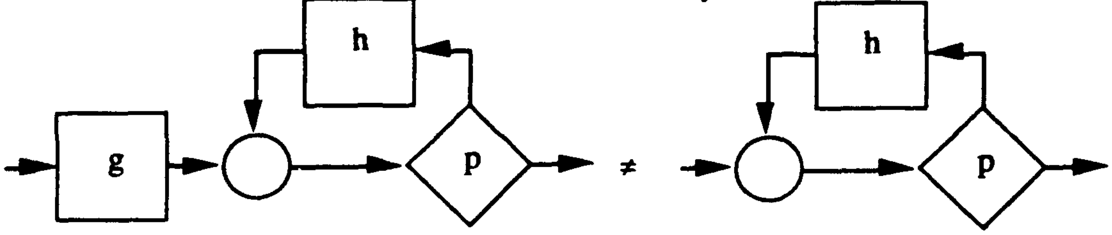
- Consider the two paths to construct final design:



The NO path makes no sense *f* ≠



and



While on the YES path, each combination of subfunctions is equivalent to the part function. The YES path represents a logical thought process. The NO path represents an illogical process. Sloppy thinking leads to sloppy programs which result in errors.

Figure 9.4.4 Useful Rules to be Followed When Writing Code

a) Keywords appear in bold, lower case print.
b) Keywords always begin a line.
c) Nested levels of code are indented two spaces more than previous levels of code
d) Constant names are in upper case print
e) Variables have the first letter in upper case followed by the rest of the variable name in lower case. Variable names may be multiple words separated by an underline (the first letter of each word should be capitalized).
f) Use only four basic structures for structured programs (sequence, iteration, alternation and concurrency).
g) Use abstract data types to manipulate data (i.e., use lists and queues rather than arrays).
h) Always include function comments to describe what function is intended. Write the function before expanding the function into code.

Some practical rules for structured programming are as follows:

a) The first abstraction is the most critical and the most difficult. It must be thoroughly thought out before it is determined. The abstraction needs to be one that can be easily verified to the problem specification.
b) Write design and code simply
c) Write design and code in an incremental manner
d) Refine each abstraction into an more detailed abstraction that is easy to prove. If it is not easy to prove, then redesign it.
e) First develop complete programs that are provable and then refine them to achieve efficiency, operation on the target processor, etc.
f) Never be afraid to start over. It will save a lot more time than trying to make a mediocre abstraction correct.
g) The last design is the best one, not the first.

9.5 Structured Programming Language Verifications

Functional verification is a procedure that enables software engineers to:

a) Write programs,
b) Read programs,
c) Verify the correctness of a program,
d) Design programs,
e) Read designs, and
f) Verify the correctness of a design,

by exploiting the fact that a program is a rule for a mathematical function. The correctness theorem, which is the formal rule for functional verification, appears in Figure 9.5.1.

Figure 9.5.1 The Correctness Theorem

For any function f and program P, correctness is defined by a condition C for

1. Complete correctness $(f=[P]) \leftrightarrow (\text{term}(f,P) \wedge f=\{(X,Y) \mid C(X,Y)\})$
2. Sufficient correctness $(f \subset [P\}) \leftrightarrow (\text{term}(f,P) \wedge f \subset \{(X,Y) \mid C(X,Y)\})$

Where P and C are:

P	**C(X, Y)**
Case (sequence): g;h	 $Y = h \cdot g(X)$

Case (ifthenelse):
if p **then** g **else** h **fi**
	
	$(p(X) \rightarrow Y = g(X)) \wedge$
	$(\sim p(X) \rightarrow Y = h(X))$

Case (whiledo): while p do g od	 $(p(X) \rightarrow Y = f \cdot g(X)) \wedge$ $(\sim p(X) \rightarrow Y = X)$

Case (fordo):
for i:∈ L(1:n) **do** go **od**
($g_{L(k)}$ is the function of the kth dopart iteration)
	
	$Y = g_{L(n)} \cdot \ldots \cdot g_{L(1)}(X)$

Case(ifthen):
if p **then** g **fi**
	
	$(p(X) \rightarrow Y = g(X)) \wedge$
	$(\sim p(X) \rightarrow Y = X)$

Case (case):
case p **part** (CL1) g ...
part (CLn) h **else** t **esac**
 	
(CL short for caselist)
	
	$(p(X) \in CL1 \rightarrow Y = g(X)) \wedge$
	...
	$(p(X) \in CLn \rightarrow Y = h(X)) \wedge$
	$(p(X) \notin (CL1, \ldots, CLn) \rightarrow Y = t(X))$

Case (dountil):
do g **until** p **od**
	$(p \cdot g(X) \rightarrow Y = g(X)) \wedge$
	$(\sim p \cdot g(X) \rightarrow Y = f \cdot g(X))$

Case (whiledo): **do1** g **while** p **do2** h **od**	 $(p \cdot g(X) \rightarrow Y = f \cdot h \cdot g(X)) \wedge$ $(\sim p \cdot g(X) \rightarrow Y = g(X))$

Note that term (f,P) is always true for loop-free programs but this formulation permits a unified treatment of loop-free and looping programs alike.

There is quite a bit of notation in this section that needs to be explained. The o in the sequence case is a mathematical composition (i.e., h o g(X) means 'h follows g(X)'). The ~ in the ifthenelse case (and also in others) represents the negation of a statement (i.e., ~ p(X) means 'when p(X) is not true'). The epsilon in the fordo case means 'is an element of.'

The theorem is stating the following:

> An intended function f is equal to (or a subset of) the function of the program P if and only iff and P have the same termination conditions and the function f equals (or is a subset of) the set of ordered pairs (X,Y), where X represents the input states and Y represents the output states, given the condition C(X,Y) is true. The conditions C(X,Y) for each type of structure that can appear in structured code is listed.

The formal rules can also be stated in the form of working questions, as shown in Figure 9.5.2.

Figure 9.5.2 Working Questions for Program Correctness

For every possible argument required by a program specification:

Case (sequence):

```
sequence = [firstpart;
              secondpart]
```

Does sequence equal firstpart followed by secondpart?

Case (ifthenelse):

```
ifthenelse = [if
                iftest
              then
                thenpart
              else
                elsepart
              fi]
```

When iftest is true does ifthenelse equal thenpart?
When iftest is false does ifthenelse equal elsepart?

Case (whiledo):

```
whiledo = [while
             whiletest
           do
             dopart
           od]
```

Is loop termination guaranteed for any argument of whiledo?
When whiletest is true does whiledo equal do part followed by whiledo?
When whiletest is false does whiledo equal identity?

Case (fordo):

```
fordo = [for
           indexlist
         do
           dopart
         od]
```

Does fordo equal firstpart followed by secondpart ... followed by lastpart?

Case (ifthen):

```
ifthen = [if
            iftest
          then
            thenpart
          fi]
```

When iftest is true does ifthen equal thenpart?
When iftest is false does ifthen equal identity?

Case (dountil):

```
dountil = [do
             dopart
           until
             untiltest
           od]
```

Is loop termination guaranteed for any argument of dountil?
When untiltest after dopart is false does dountil equal dopart followed by dountil?
When untiltest after dopart is true does dountil equal dopart?

In order to prove the correctness of programs, the strategy is to use the techniques, specifically:

a) Know the proof rules
b) Read and abstract programs
c) Hypothesize and substitute functions
d) Reach informed correctness conclusions
e) Conduct team proof reviews.

Programs are to be written for verification. Specifically, a program needs to be refined in a manner that will make verification as easy as possible. There will be a series of refinements, leading from the Clear Box to the code. Making each refinement small will also make it simpler, easier to understand and easier to verify.

The formal proofs are done using the trace table notation, where the function of each line of code is reflected in modifications to variables and conditions. An example of a trace table for a segment of code appears in Figure 9.5.3.

An extra moment must be spent explaining the iteration structure (the while loop). Conceptually, a terminating while loop that will terminate in at most N iterations can be represented by a sequence of N alternation (if-then) structures. In order to prove the correctness of a while loop, one must first prove termination, and then prove that the while loop is equivalent to some sequence of if-then structures. This concept, known as the Iteration Recursion Lemma, is more formally defined and illustrated in Figure 9.5.4. A trace table for a while-loop is shown in Figure 9.5.5. In this trace table, line 2 represents the first if-then structure. The third line represents the effect of zero or more repetitions of the if-then structures by the Iteration Recursion Lemma

In verifying programs, a balance between formal procedures and economy of effort must be reached. This can be achieved by realizing that large programs have simple parts that:

a) For the most part are better verified by direct assertions, and
b) Lead to the use of formal verification on an exception basis.

Any correct program can be verified. If it is unstructured, it may need first to be structured; but verification is always possible. Unfortunately, programs not written using stepwise refinement with functional verification will often be extremely difficult to verify. This occurs because most of these programs were not written to be verified; that is, they were not written in a simple, easy-to-read manner, using consistent refinements. Corrections made during debugging also make the code less readable. Programs written using the concepts of stepwise refinement and functional verification are relatively easy to verify. Most programs can be verified by direct assertion, without the need to do a detailed and complicated formal verification.

Some practical rules for verification are as follows:

a) Code should be provable by direct assertion.
b) Code that must be proven formally should have a proof that is easy to understand.

Figure 9.5.3 Trace Table for Sequence Structure

- T race table:

	part	w	x	y	z
1	w := x + y	$w_1 = x_0 + y_0$			
2	x := y + z		$x_2 = y_1 + z_1$		
3	y := z + w			$y_3 = z_2 + w_2$	
4	z := w + x				$z_4 = w_3 + x_3$
5	w := y - z	$w_5 = y_4 - z_4$			
6	x := z - w		$x_6 = z_5 - w_5$		
7	y := w - x			$y_7 = w_6 - x_6$	
8	z := x - y				$z_8 = x_7 - y_7$

- Derivations:

$$\begin{aligned} w_8 &= w_5 \\ &= y_4 - z_4 \\ &= y_3 - w_3 - x_3 \\ &= z_2 + w_2 - w_2 - x_2 \\ &= z_2 - x_2 \\ &= z_0 - y_1 - z_1 \\ &= z_0 - y_0 - z_0 \\ &= -y_0 \end{aligned}$$

$$\begin{aligned} x_8 &= x_6 \\ &= z_5 - w_5 \\ &= z_4 - y_4 + z_4 \\ &= 2z_4 - y_4 \\ &= 2w_3 + 2x_3 - y_3 \\ &= 2w_1 + 2x_2 - z_2 - w_2 \\ &= 2w_1 + 2y_1 + 2z_1 - z_1 - w_1 \\ &= w_1 + 2y_1 + z_1 \\ &= x_0 + y_0 + 2y_0 + z_0 \\ &= x_0 + 3y_0 + z_0 \end{aligned}$$

Etc.

- Program function:

$w, x, y, z := -y \;, x + 3y + z, -x - 4y - z, 2x + 7y + 2z$

c) If proving a code unit formally is complicated or difficult to prove, then re-design and re-code the code unit.

d) When a verification is complete, there are two additional conditions that will help validate the correctness of the proof. Those conditions are:
   1) Is the entire domain for the function covered (i.e., is the function complete)?
   2) Are all calculations and manipulations correct (i.e., is the function correct)?

e) Use functional commentary to define what the refinement will do, and what it has done.

Figure 9.5.4 Iteration Recursion Lemma

- **Iteration Recursion Lemma.**

  Given functions f, g, h, and predicate p

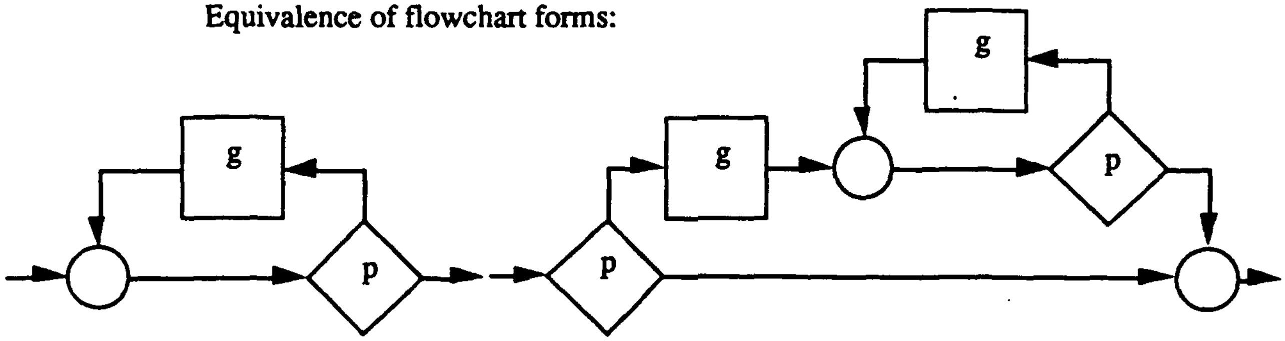
  $(f = [\ \textbf{while}\ \ p\ \ \textbf{do}\ \ g\ \ \textbf{od}\ ])$ ′

  $(\text{term}(f,\ \ \textbf{while}\ \ p\ \ \textbf{do}\ \ g\ \ \textbf{od}\ ) \wedge$

  $f = [\ \ \textbf{if}\ \ p\ \ \textbf{then}\ \ \ g; f\ \ \ \textbf{fi}\ ])$

- Proof idea (whiledo)

  Equivalence of flowchart forms:



Section 9.6 Conducting Proof Reviews

In addition to verifying the correctness of any level of design or code individually, the proof must then be reviewed by the entire Development team. This is to minimize the probability of human fallibility, since a number of different people will look at the same item. Additionally, the review confirms the correctness of the proof, which in turn, proves the correctness of the design.

The manner in which reviews are conducted are dependent on the organization, project, team and invention to be reviewed. The frequency with which reviews are conducted are also dependent on the project and invention to be reviewed. For example, in some projects, the team may determine that there is no need to pass out material to be reviewed before the review, because there will be a walkthrough as a part of the review, and if any part of the refinement is too complicated to be understood immediately, it should be redesigned. This may be an extreme example, but illustrates the need for review formats to be organization/ project specific. In another organization, where reviews of small inventions are held daily, there may not any need to distribute the material to be reviewed more than a few hours in advance. Issues pertaining to the proof review process must be carefully considered in order to create a proof review process that is efficient and effective.

A workable procedure for organizing for the proof reviews process is: prior to a proof review, the person whose material is to be reviewed distributes the material to be reviewed, along with the material to

Figure 9.5.5 While-loop Trace Table

function (x, y, and a integers)

$f = (x \geq 0 \rightarrow x, y, a := 0, a*x + y, a)$

program

```
1  while
2      $x \neq 0$
3  do
4      $x, y := x - 1, y + a$
5  od
```

proof

term

Initial ≥ 0 is reduced by 1 each iteration, so eventually whiletest $x \neq 0$ fails

pass

whiletest true (prove $(p \rightarrow f) = (p \rightarrow f \circ g)$)

	<u>part</u>	<u>condition</u>	<u>x</u>	<u>y</u>
2	$x \neq 0$	$x_0 \neq 0$	$x_1 = x_0$	$y_1 = y_0$
4	$x, y := x - 1, y + a$		$x_2 = x_1 - 1$	$y_2 = y_1 + a_0$
f	$x, y, a := 0, a*x + y, a$	$x_2 \geq 0$	$x_3 = 0$	$y_3 = a_0*x_2 + y_2$

conditions:

$$
\begin{aligned}
x_0 \neq 0 \wedge x_2 \geq 0 &= x_0 \neq 0 \wedge x_1 - 1 \geq 0 \\
&= x_0 \neq 0 \wedge x_0 - 1 \geq 0 \\
&= x_0 \neq 0 \wedge x_0 \geq 1 \\
&= x_0 > 0
\end{aligned}
$$

assignments:

$$
\begin{aligned}
x_3 = 0 \quad y_3 &= a_0*x_2 + y_2 \\
&= a_0*(x_1 - 1) + y_1 + a_0 \\
&= a_0*(x_0 - 1) + y_0 + a_0 \\
&= a_0*x_0 + y_0
\end{aligned}
$$

program function: $(x > 0 \rightarrow x, y, a := 0, a*x + y, a)$

which agrees with intended function when whiletest true.

pass

whiletest false (prove $(\sim p \rightarrow f) = (\sim p \rightarrow I)$)

$(x = 0 \rightarrow (x \geq 0 \rightarrow x, y, a := x, a*x + y, a))$

$= (x = 0 \rightarrow x, y, a := 0, a*0 + y, a) = (x = 0 \rightarrow x, y, a := x, y, a)$

that is, the identity function, as required.

pass

result pass comp

put the area of review into context, a sufficient amount of time before the review is to take place. In that manner, all of the developers participating in the review have the opportunity to re-familiarize themselves with the material. Depending on the organization and schedule, a sufficient time may be a few minutes or a few days. At the review, the developer who created the material walks through the design or coding, with comments being recorded by one of the other developers. Corrections may be suggested and action items may be assigned. If numerous or non-trivial errors were found, then another review must be conducted. Otherwise, the material does not need to be reviewed until it is refined again.

Section 9.7 Working With The Certification Team

The lines of communication between the Development and the Certification team are extremely clear and simple. The Development team delivers code to the Certification team, and receives failure reports from the Certification team. Of course, there are additional communications, such as the Certification team stating that a code increment was successfully certified or that a certain increment must be re-developed, but the two above-mentioned categories are the major ones. As the Development team completes the coding and verification of an increment, the code is then delivered to the Certification team so it may be placed under configuration control. If the code has been modified, then an Engineering Change Notice must be attached to it. When failures are found in the code, be they compilation, linking or execution failures, failure reports are filled out and given to the Development team so they may be resolved.

There are a few key points to be clarified here:

a) Certifiers do not modify the code. In fact, they have no reason to actually look at the source code.
b) All changes (due to failures, spec changes, enhancements, etc.) to the code are made by the Development team.
c) Any enhancement (changes not due to failures) must be justified to a configuration control board.

Section 9.8 Working With The Specification Team

The Development and Specification team work together to develop the Functional (Black Box) Specification Volume, and may interact when the Construction Plan is written. During actual development or certification, the Development and Specification teams will only interact when the specifications have been changed or when a request to change the specifications is made. When the specifications are changed, the Specification team will publish a new version of the software specifications so that the Development (and Certification) team will be aware of the new function for which a rule must be written and certified. If the Development team determines that some part of the specifications may need to be changed, they submit a request, in the form of a question or issue, which will be then be studied by the Specification team. The result will be a response to the question or issue and, if the modification is deemed correct, a new set of software specifications. It is vitally important that the Development team add nothing to the design or code that is not in the specifications. For this reason, the Development team must wait until the specification change request is approved before design or code reflecting that change is created.

Section 10
Cleanroom Certification Team Practices

In this section, the Cleanroom Project cycle is examined from the vantage point of the Certification team. In the Cleanroom environment, the Certification team has the following well-defined mission:

> Given a system of programs which has been implemented and verified, conduct measurements to determine how well the program satisfies the specification in such a way that the reliability of the software can be certified.

In performing the measurements and certification, the Certification team is making use of the fact that software use is stochastic. Therefore, if random samples of possible usages are drawn from the population of the uses that the software will be subject to in performing its intended mission, statistical theory can be used to estimate the reliability of the software.

In performing this mission, the Certification team is performing the engineering tasks defined in the processes below

E19: Develop Certification Tests for Increments 1...j
E22: Build System With Increment i
E23: Certify Increment 1...j.

The Specification team is responsible for developing and refining the specification (Process E14), but members of the Certification team may well contribute to the development of Volume V - The Expected Usage Profile. For this reason, the useful techniques for the development of usage profiles are discussed in this section.

The Certification team also contributes to performing the work defined by the three processes that have to do with understanding the problem and solution domains:

E12: Understand Problem Domain
E13: Understand Solution Domain
E21: Increase Understanding of Problem and Solution Domains as Required

The Certification team's principal effort is devoted to accomplishing the previously referenced processes (E19, E22 and E23). Therefore, this section is devoted to discussing their performance of the engineering tasks associated with these processes. In addition, the Certification team contributes to performing the project overhead tasks as required. These include:

E7: Maintain Project Schedule
E8: Prepare for and Conduct Project Review
E9: Prepare and Submit S.atus Reports
E11: Submit a Question or Issue

The Certification team is provided with a specification that contains all the information they require to prepare tests. They are given the expected usage profile in the form of a Markov Model, and they have all the informatic n required to develop test cases in Volume IV - User's Reference Manual. The Certification team requires no knowledge of the rule (program) since the test regime used in the Cleanroom environment is usage testing.

The Certification team develops a test plan and sampling plan which will guide them in determining how many test cases are required for certifying the software, which is performed in process E19: Develop Certification Tests for Increments 1...j. The Certification team can develop test cases for increments 1...j in parallel with the Development team's design and implementation of increment j.

The Certification team is responsible for configuration management of the software system as it is being developed. Therefore, when the Development team turns over a verified increment, they compile and otherwise perform the tasks necessary to develop an executable system. They run test cases and when a failure is observed, they prepare a Software Failure Report. Eventually, when the Certification team determines further testing is of marginal value, the accumulated Software Failure Reports are returned to the Development team for resolution. The Certification team then waits for Engineering Change Notices, which contain descriptions of the fixes to the observed failures, along with the modified.

The Certification team stores all failure data in a data base for analysis and later use by the certification model. The certification model is run for each software system version in order to predict the reliability of the next version of the system. The Certification team keeps the Engineering team manager informed of certification progress. At each testing stage, the Certification team can make one of three recommendations: (1) continue certification, (2) the desired reliability has been obtained so terminate certification, or (3) the software for the increment as designed and implemented is faulty so it is best to discard the current implementation and begin with a clean slate. A useful rule of thumb for making this latter determination is if any deep design failure has been found which requires changes to any significant amount of code or state data or if more than five failures per KLOC have been found, the current software should be discarded and a new development effort initiated. If testing is continued, it will be necessary to start testing in quality with all the associated hazards that this strategy entails.

All of these points will be discussed in greater detail in the following subsections:

10.1 Preparing the Sampling Plan

The sampling plan organizes the manner in which the certification for an increment is performed. The Certification team must determine whether there will be stratified testing, and how the testing resources will be divided. For example, if there is a safety critical portion of a software system, it may be desirable to determine a reliability for that subsystem, in addition to the reliability for the entire system. Additional issues that must be considered when developing the sampling plan include the following:

a) Is control flow to be randomized?
b) Are data inputs to be randomized?
c) What is the expected failure rate?
d) What is a use?
e) What is the average size of a test case?
f) What are the units used in order to determine MTTF?
g) What is the desired reliability?
h) What is the time period available for the certification of the increment?
i) What are the resources available for the certification of the increment?

In preparing the sampling plan, the large body of knowledge that has been developed relative to sampling should be considered to obtain the best results with the minimum of effort. For the statistical inferences made based on the sample to be valid, samples must be chosen in such a way that they are representative of the population. Factors to consider in designing the sampling plan are:

a) size of sample
b) draw samples with or without replacement
c) how to randomize selection of sample elements
d) how to manage selection of sample elements (the sampling plan)

In developing the sampling plan, there are two general strategies:

fixed design — the design is fixed before sampling begins

sequential sampling — a small random sample is selected and analyzed and on the basis of the result, a decision is made to continue sampling and, if so, how

There are several useful **fixed design sampling strategies** as follows:

Unrestricted random sampling - a random sample is selected from the whole population by either

Simple random sampling — Assign a different number to every element of population and use random numbers to select the sample

Systematic Random Sampling — Order the population and select a random starting point and then select every n(th) element where the interval size (n) is chosen so the desired sample size is obtained

Restricted random sampling

The population is divided into groups (and perhaps the groups are also subdivided) and random samples are either drawn from each group or randomly selected groups

Multistage random sampling

Random samples are drawn from subgroups which have themselves been selected with equal probabilities, or with probabilities proportionate to the relative size of the subgroup or some other criterion

Stratified random sampling

Random samples are drawn from every subgroup of the population. The size of the sample from the subgroup may be: (a) independent of the size of the subgroup (i.e., equal size), (b) proportionate to the relative size of the subgroup or (c) proportionate to the relative size of the subgroup and the dispersion of the elements in the subgroup

Cluster sampling

A random selection of subgroups is selected and all elements in the subgroup are included in the final sample

Stratified cluster sampling

A combination of stratified random sampling and cluster sampling in more than two stages

There are several useful **sequential sampling strategies**. In sequential sampling, a small random sample is selected and analyzed. The result is used to make a decision as to whether or not to continue to sample and, if so, how. The samples may either be

in groups or
single items.

Selecting the sequential sampling plan to use is based on such factors as:

the characteristics of the population
the cost of taking the sample
the cost of processing the sample
the variance of the estimates

It is very important that the sampling plan be well thought out in advance to get the best possible estimate of reliability over the entire domain of interest with the greatest efficiency.

Upon assessment of these issues, a report will be written which clearly outlines how certification will occur for this increment. It will list a strategy, which is basically a division of resources for certain parts of the problem, and expected results, in terms of reliability. This plan will then serve as the basis for test cases to be generated, executed and validated.

10.2 Preparing Test Cases: The Monte Carlo Method

The process of preparing test cases is divided into a number of activities, the most important of which is the Monte Carlo method. The activities are:

a) Incremental Development
b) The Monte Carlo Method
c) The Test Generation Algorithm
d) Random Numbers
e) Test Case Generation

Incremental development leads to the following features:

a) Software is tested by increment,
b) It allows iterative assessment of software,
c) It allows early increments to be certified more thoroughly, and
d) It makes the testing effort more manageable.

The number of test cases per increment must be determined. It is a function of:

1) Budget,
2) Schedule, and
3) Desired MTTF for the increment.

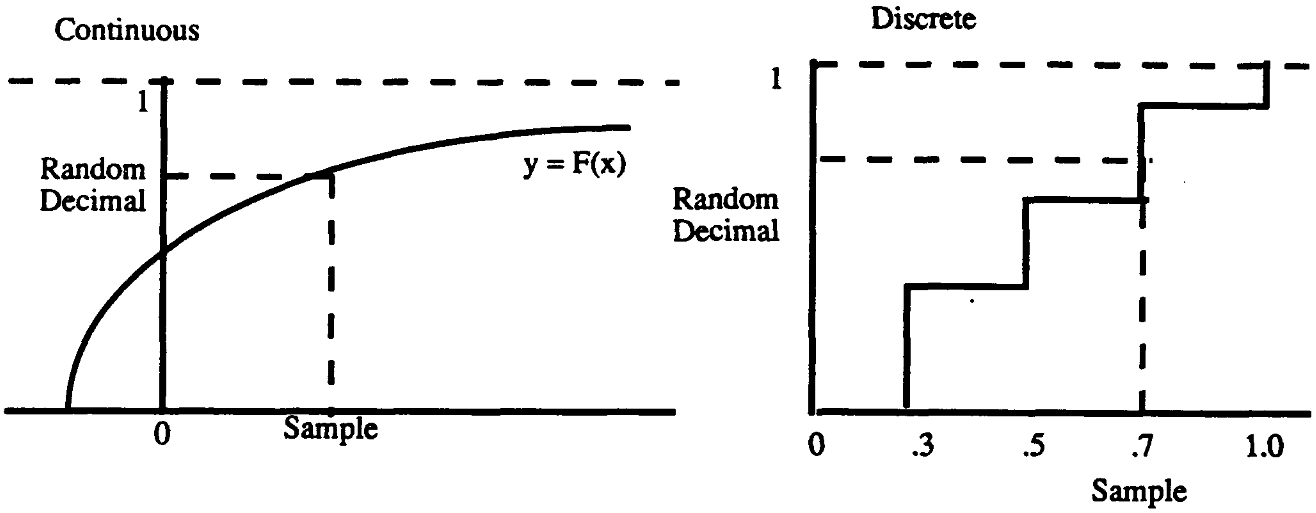
The Monte Carlo technique is a procedure to obtain approximate evaluations of mathematical expressions which are built up of one or more probability distribution functions. Monte Carlo is performed by conducting a simulation experiment to determine some probabilistic property of a population of objects or events by the use of random sampling applied to the components of the objects or events.

The Monte Carlo technique works as follows:

a) Plot the cumulative probability function (y axis will go from 0 to 1). The function can be discrete or continuous.
b) Choose a random decimal between 0 and 1, which is the y value (to as many decimal places as necessary/desirable). Find this point on the cumulative function.
c) Determine corresponding x axis point.
d) The value of x is the sample value of x.

The Monte Carlo technique is illustrated in Figure 10.2.1 for the continuous and discrete cases.

Figure 10.2.1 Continuous and Discrete Monte Carlo Method



The test case generation algorithm is given below:

```
start in initial state
while test case not complete
    do
        Randomly select a state transition from the present state using the Monte Carlo method
        and the probabilities in the transition matrix for the row of the initial state.
        Append the script for the state transition to the test case script.
            if script includes randomly generating inputs according to their usage profile that are
            necessary for that script
            then
                generate them
        fi
        Move to the row of the new state
            if this state is the final state OR test case is of determined length (which can also be
            randomly determined)
            then
                test case complete
        fi
    od
```

Test paths are randomly generated according to state transition probabilities. Input are generated according to the probability of use. This permits the statistical generation of stimuli (e.g., keystrokes, sensor readings) and statistical interpretation of testing results. Figure 10.2.2 shows a fragment of a program designed to conduct a Monte Carlo simulation to generate test scripts at random. Figure 10.2.3 shows the test script fragment generated by the program fragment shown in Figure 10.2.2. These fragments are then combined to form full test scripts. In generating test scripts, it is sometimes more efficient from the standpoint of test evaluation to remove the variability of test inputs by fixing them. This eases validation of test results, but biases the approach somewhat, since randomness is only in terms of control flow. One should always try to have variability in control flow and input.

Figure 10.2.2 A Typical Script Generation Rule

```
  .
  .
  .
x = ran(0)
if x > .5
  then
    write "Move A west"
  else
    write "Move A east"
fi
write "Move B to A"
x = ran(1)
if x > .25
  then
    write "Put new train at D"
  else
    write "Put new train at A"
fi
  .
  .
  .
```

Figure 10.2.3 A Typical Test Script Fragment

```
State 6 to State 8 -
        Move A west
        Move B to A
        Put new train at D
```

The test cases themselves are:

- Based on transition probability (Markov) matrix,
- Some likely sequences may not be selected due to the fact that state transitions and inputs are selected randomly, even though they are selected in the same proportion they are likely to occur in actual usage,
- The present state tells us which stimuli are possible, and their probabilities, and
- The stimulus selected will direct us to the correct new state (which may be the same).

A test script is a list of the combination of stimuli which lead to a state transition. The selection of a particular script is based on the state transition selected. A test case can be made up of multiple state transitions. Therefore, it can be made up of a number of test scripts. Inputs in a test script may be selected randomly, but other stimuli should be identical for all invocations of a particular state transition. For example, if two particular buttons need to be pushed for a specific state transition, those buttons are pushed.

When a random number is generated, the stimulus corresponding to that number while in a specific state is put into the test script. The system should now be in the next (maybe same) state. The stimuli can be keystrokes, sensor readings or input data. The next stimulus is selected by generating another random number and finding the stimulus corresponding to it while in the new state. In this way, one is constantly traversing the transition matrix to keep track of the present state. An example of a test case appears in Figure 10.2.4.

Figure 10.2.4 A Test Case

Test Case - Examples

Test Case : 3 Pass____ Fail____

```
119   1. S1_Initial
120   2. S1_Down_Arrow
121   3. S1_Enter_Certification_Model
122   4. S5_Initial
123   5. S5_F3
124   6. S5_F1
125   7. S1_Initial
126   8. S1_Down_Arrow
127   9. S1_Down_Arrow
128  10. S1_Enter_Exit
```

10.3 Running Test Cases

The script for a test case is generated by the Markov Transition matrix and a random number generator. The test cases are keyed in by the tester exactly as the script is written. The one checks if the result is the expected result. Mistakes in pressing the keys are acceptable, since it is just a random change in the test script. Intentional or planned mistakes should not occur, since that removes the randomness from test case. The software must still work in the desired manner given any stimuli, correct or incorrect.

Some portion of the stimuli can be set aside for random blunders, say 0.5%. These blunders include: hitting neighboring keys and taking imperfect program paths. The blunders are planned during the test case generation.

Results of test cases need to be maintained. A typical form on which solutions can be manually recorded is shown in Figure 10.3.1. The exact format will depend on the software being certified. In many instances, it is worth the effort to develop a means to record the information in an electronic file so it can be easily compared to the expected results.

Figure 10.3.1 Test Solution

Test Scenario Number: ___________

Enter the expected result in the space provided, or attach additional sheets that clearly describe which state transition the expected result is for.

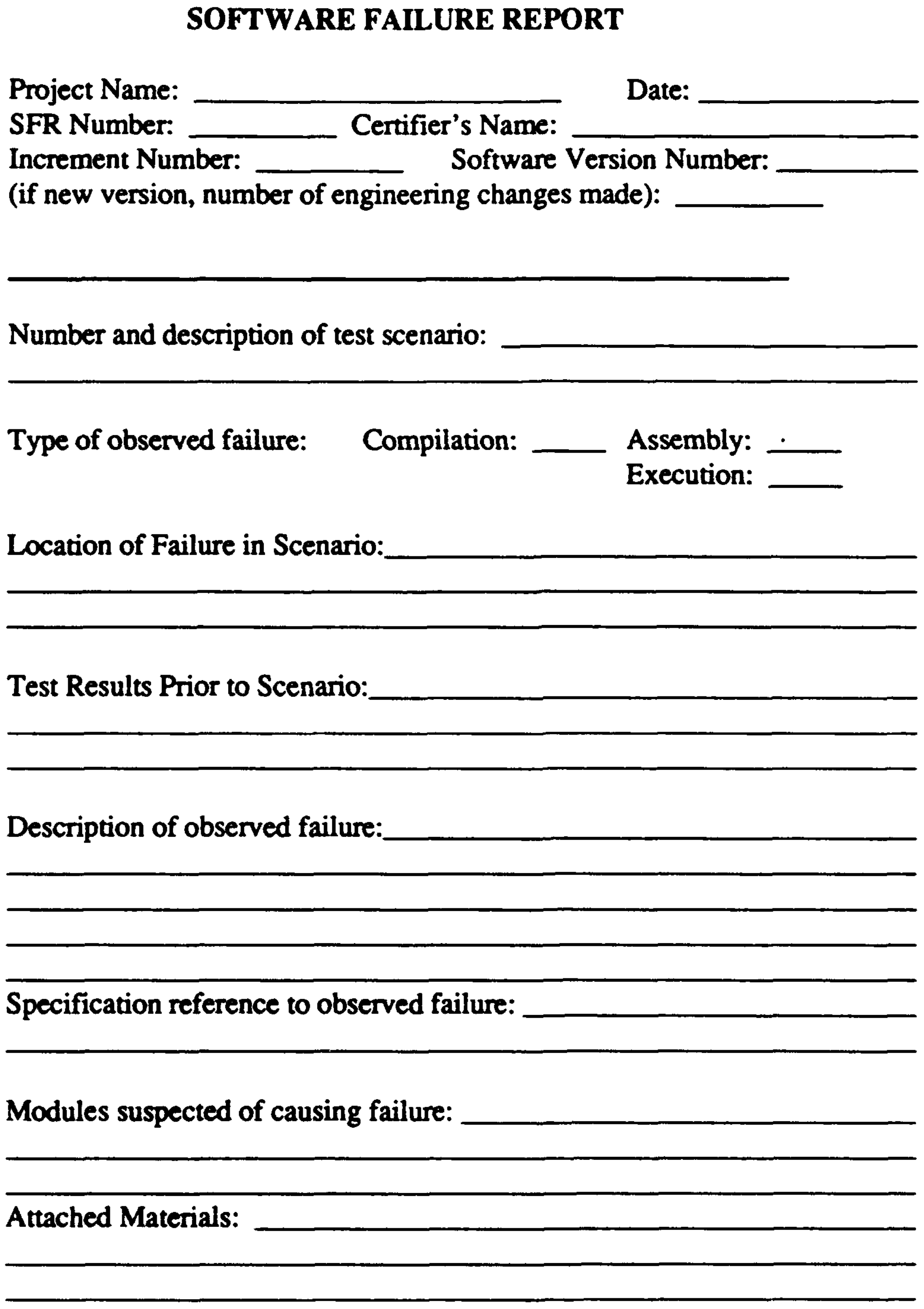
State Transition 1 Result: ______________________________
State Transition 2 Result: ______________________________
State Transition 3 Result: ______________________________
....
....
Expected Result: ______________________________

10.4 Testing Results and Analysis

When a test case is executed, two types of results appear. One is the program output, which is the responses generated by the program being certified. The second is the certification data, which is the information necessary to run the certification model.

Once the program output is generated, where screens are written to, text is printed, etc., the certifier must determine whether the test has executed correctly. To do this, the expected result is compared to the actual result. If an inconsistency is found, and the test case failed, a failure report is prepared and submitted to the state data file. A typical failure report is described and illustrated in Figure 10.4.1.

Figure 10.4.1 Software Failure Report



SOFTWARE FAILURE REPORT

Project Name: ____________________ Date: ____________

SFR Number: ________ Certifier's Name: ________________

Increment Number: ________ Software Version Number: ________

(if new version, number of engineering changes made): ________

Number and description of test scenario: ______________________

Type of observed failure: Compilation: _____ Assembly: _____
Execution: _____

Location of Failure in Scenario: ______________________

Test Results Prior to Scenario: ______________________

Description of observed failure: ______________________

Specification reference to observed failure: ______________________

Modules suspected of causing failure: ______________________

Attached Materials: ______________________

For every test case executed, a body of information must be gathered in order to run the certification model. The system version, test case executed (and whether it was the first time the test case was executed), the execution time, whether the test case failed, failure report number, and type of failure (compilation, linking or execution) must be recorded. In this way, the reliability of the system can be determined in a variety of ways (with compilation failures, without counting failures from retrograded tests, etc.). Since a large number of categories are being handled for each test case, and there are a variety of ways in which this data should be analyzed, storing this information in a relational data base is useful.

When enough failures have accumulated, failure reports can be given to the Development team so they may correct the failures. If failures have not been found, and the software system has reached its reliability target, the certification of the next increment can begin.

10.5 Working With The Certification Model

Actual reliability results are entered into the Reliability Manager in order to project reliability for the next increment or software system version. Additionally, actual reliability results can be compared to projections in order to determine the need for additional testing or redevelopment.

The certification model provides an actual assessment of the reliability of the product in terms of its operational use. In that manner, one can clearly project how the software system will work in the real world.

The Cleanroom Reliability Manager (CRM) is a tool that has been created in fulfillment of another portion of this task. Given test execution results (version number, execution time, pass or fail), the CRM tool is used to calculate a projected reliability for the next version of the software system in terms of Mean Time to Failure. In the case where the final test scenarios were executed and a failure was not found, the execution time to be entered into the model for that series of scenarios will be double the execution time. There is a min-max argument that justifies this correctness of this extrapolation. The certification model is an exponential growth model, which is based on the fact that selecting test cases according to the operational profile will tend to find the most frequent failures first. When frequent failures are removed, the Mean Time to Failure will grow at an exponential rate. The power of testing according to a model is that it will give a more accurate projection of the future reliability of the software system.

Certification can stop as a result of one of two conditions. First, the reliability goal for the increment may have been reached. In this case, the certification of the next increment may begin. Second, the failure density may be above the threshold to continue testing. Five failures per KLOC is considered to be a fair failure density limit. In this case, the increment should be redesigned, as it will cost less to start over again than it will to continue to try to test quality into the product. Otherwise, certification should not be considered complete, which means additional test scenarios may need to be generated randomly.

10.6 Configuration Management

Since developers cannot compile (let alone test) their code, the Certification team is responsible for all configuration management. The certifiers must log in all new or changed components, while specifically keeping track of the change history of each component. This means all modified components need to have an engineering change notice attached. Engineering change notices are described and illustrated in Figure 10.6.1. All components must then be compiled and assembled. Failures found during compilation and assembly result in failure reports being submitted and a corresponding entry being entered with the rest of the certification data.

If there are software initiation procedures, they must also be followed. Only then can test cases be executed on the code.

10.7 Working With the Development Team

The lines of communication between the Certification and the Development teams are clear and simple. The Certification team receives code from the Development team, and returns failure reports to the Development team. Of course, there are additional communications, such as the Certification team stating that a code increment was successfully certified or that a certain increment must be re-developed, but the two above-mentioned categories are the major ones. As the Development team completes the coding and verification of an increment, the code is then delivered to the Certification team so it may be placed under configuration control. If the code has been modified, then an Engineering Change Notice must be attached to it. When failures are found in the code, be they compilation, linking or execution failures, failure reports are filled out and given to the Development team so they may be resolved.

There are a few key points to be clarified here:

a) Certifiers do not modify the code. In fact, they have no reason to actually look at the source code.
b) All changes (due to failures, spec changes, enhancements, etc.) to the code are made by the Development team.
c) Any enhancement (changes not due to failures) must be justified to a configuration control board.

10.8 Working With the Specification Team

The Certification and Specification team work together to develop the Usage Profile Volume and may interact when the Construction Plan is written. During actual development or certification, the Certification and Specification teams only interact when the specifications are changed, or when a request to change the specifications is made. When the specifications are changed, the Specification team publishes a new version of the software specifications so that the Certification (and Development) team are aware of the new function for which a rule must be written and certified. If the Certification team determines that some part of the specification needs to be changed, they submit a request in the form of a question or issue, which

Figure 10.6.1 Engineering Change Notice

ENGINEERING CHANGE NOTICE

Developer's Name: ____________________ Date: __________
ECN Number: ________ Authorizing signature:______________
Type of Resolution:
_______ failure corrected
_______ change implemented
_______ no change required
_______ other (Describe):
____
____

To be completed if ECN is a failure or change
Failure Reports fixed by this ECN: ____________________

Textual description of ECN:______________________
____
____
____
____
____
____

Routines modified by ECN:______________________
____
____
____
____

Changes made to each routine (attach additional sheets if necessary): ___
____
___ ____________________________________

Source of modification (check one):
Requirements:____ Specifications:____ Black box design:____
State box design:____Clear box design:____ PDL:____
Code:____ Previous modification:____ Other:____

will be studied by the Specification team. The result will be a response to the question or issue and, if the modification is deemed correct, a new set of software specifications.

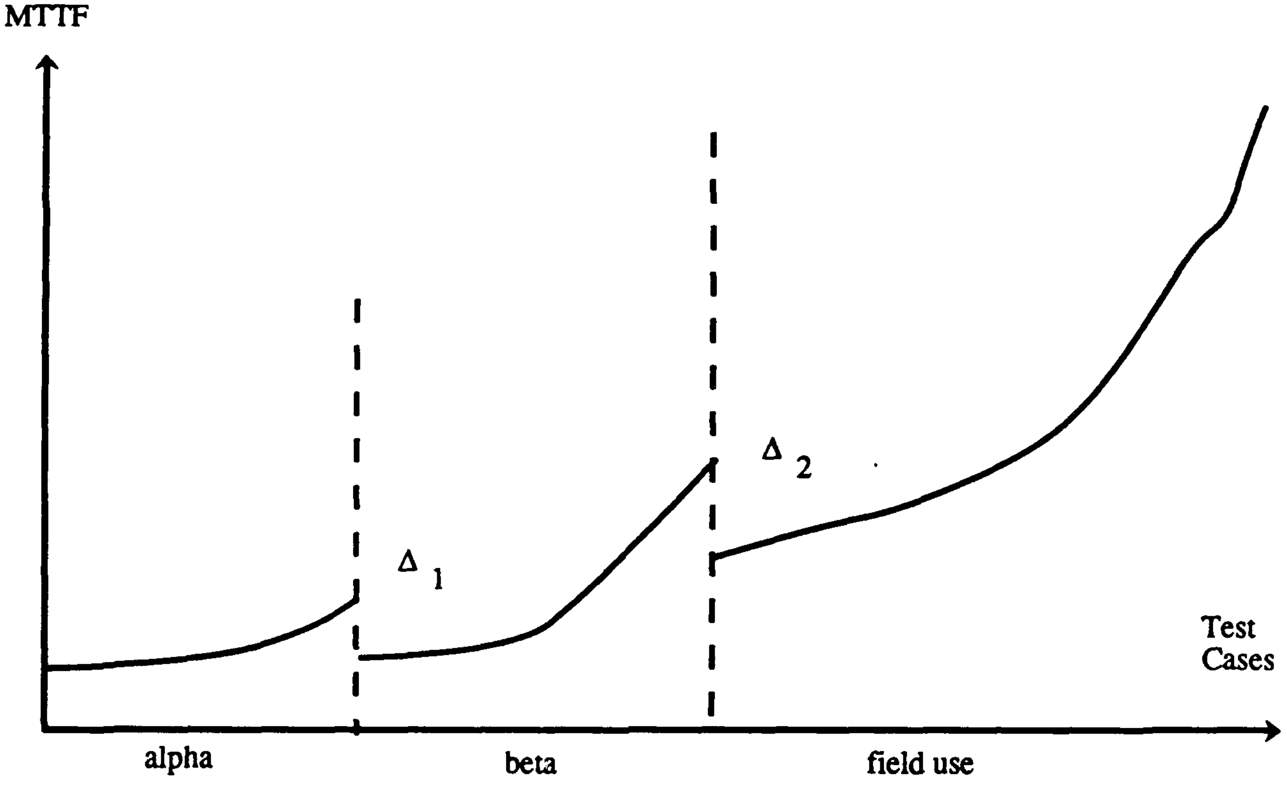
10.9 An Integrated Program of Usage Testing

For most projects, the testing budget is limited. It may be possible that a sufficient budget is not available to certify the reliability of the software system created to the desired level. Additionally, an organization may not be sure whether the usage profile that the software was certified is correct. In these cases, it is desirable to have a number of levels of testing.

First, the Certification team does testing in the laboratory, certifying the reliability of the software to the limit that the sampling strategy or budget allow. Once that is completed, the software is distributed to a number of actual users of the system. This level of testing is called beta testing. These users may be inside or outside the organization. They are using the system with the knowledge that this is a pre-release version of the software. The software is then distributed to a limited number of clients in the field. These organizations will use the software in their day-to-day tasks. The incentives for organizations to be beta or limited field-use test sites is a management issue. Finally, the software is distributed for full field use, where the reliability of the software will have been certified to some level, based on the laboratory, beta and limited field-use testing.

The objective of this three-level certification process is to certify the software to the highest possible level at the lowest possible cost. In this manner, the software product is tested thoroughly with three different usage profiles (although ideally they will be the same). During the laboratory, beta and limited field use testing processes, execution time and failures must be returned to the Certification team so the Development team may correct the higher frequency failures and for the certifiers to project the future reliability of the system. Extremely low frequency failures typically may not be corrected if the probability of creating a new error is considered to be higher than the probability of making the right correction. The reliability projections for a software system using this three-pronged approach will typically appear as shown in Figure 10.9.1. The reliability of the system will typically decrease at the commencement of a new level of testing. This is a result of the new or different usage profile. But as those failures are resolved, the reliability should increase exponentially.

Figure 10.9.1 Reliability Projections for a Three-Pronged Testing Approach



An efficient testing strategy will minimize Δ_1 and Δ_2

APPENDIX A:

This appendix contains the list of completion conditions for each of the Cleanroom processes. The forms are suitable for distribution to the engineering teams with task assignments. In this way the team can use the form to record progress and report completion. The form asks for signatures of the entire team to make clear the team responsibility for successfully completing the process.

Completion Condition Checklist:

Project: ____________________ **Process:** E0: Cleanroom

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Have all necessary measurements been made?
	2. Have all completion conditions for all six sub-processes (E1, E2, E3, E4, E5 and E6) been achieved?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________________ **Process:** E1: Project Invocation

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Have all initial staff members been put on a team?
	2. Has the initial Project Charter been accepted?
	3. Has the software development plan been created and accepted?
	4. Has the initial project schedule been created?
	5. Have initial task assignments been given?
	6. Have all additions to, deletions from and modifications of the state data been completed?
	7. Have all necessary measures been gathered?
	8. Have all necessary responses been submitted?
	9. Have all pertinent reviews for this process been completed?
	10. Have all action items generated during reviews that pertain to this process been completed?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: **Process:** E2: Program Management

Engineering Team Manager: **Iteration #:**

Date Completed:	Completion Condition
	1. Have all necessary measures been gathered?
	2. Have all completion conditions for all three subprocesses (E7, E8 and E9) been achieved?
	3. Is the project completed?

Team members' signatures

Manager's signature

Completion Condition Checklist:

Project: ____________________ **Process:** E3: Project Information Management

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Have all necessary measures been gathered?
	2. Have all completion conditions for both subprocesses (E10 and E11) been achieved?
	3. Is the project completed?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________________ **Process:** E4: Software Solution Specification

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Have all necessary measures been gathered?
	2. Have the completion conditions of all four sub-processes (E12, E13, E14, and E15) been achieved?
	3. If the project was suspended, has all material been archived in a manner that will be useful to individuals who will need the material in the future?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________________ **Process:** E5: Software Development and Certification

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Have all necessary measures been gathered?
	2. Have all completion conditions from both subprocesses (E16 and E17) been achieved (This is equivalent to asking whether all increments have been developed, verified and certified)?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________________ **Process:** E6: Prepare Final Project Releases

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Are all deliverables complete and consistent?
	2. Have all necessary measures been gathered?
	3. Have all additions to, deletions from and modifications to the state data been completed?
	4. Have all responses from the process been submitted?
	5. Have all pertinent reviews for this process been completed?
	6. Have all action items generated during reviews that pertain to this process been completed?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________________ **Process:** E7: Maintain Project Schedule

Engineering Team Manager: ____________________ **Iteration #:** ______

Date Completed:	Completion Condition
	1. Have all defined task assignments been given a completion schedule, as well as staff and resource allocations?
	2. Are all task assignments (internal schedule) consistent with the project schedule and the Master Project Schedule (external schedule)?
	3. Have all known milestones been put on the schedule?
	4. Is the project schedule complete and consistent in relationship to the Master Project Schedule?
	5. Has the project schedule been published due to changes in the Master Project Schedule or by time interval?
	6. Does the project schedule list all sponsor requested reviews?
	7. Have all additions to, deletions from and modifications of the state data been made?
	8. Have all necessary measures been gathered?
	9. Have all external responses been submitted?
	10. Have all pertinent reviews for this process been completed?
	11. Have all action items generated during reviews that pertain to this process been completed?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________________ **Process:** E8: Prepare For and Conduct Project Review

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Has the Engineering team manager determined that the Specification, Development and Certification teams have sufficiently prepared for the project review?
	2. Have all additions to, deletions from and modifications of the state data been completed?
	3. Have all external responses been submitted?
	4. Have all necessary measures been gathered?
	5. Is the sponsor satisfied that the review is complete? If not, has another review been scheduled?
	6. Have all action items from the review been assigned?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ______________ **Process:** E9: Prepare and Submit Project Status Reports

Engineering Team Manager: ______________ **Iteration #:** ______

Date Completed:	Completion Condition
	1. Are the Project Status Reports complete and accurate for the time period for which the status report is being submitted?
	2. Are consistent criteria being used to complete the Project Status Reports?
	3. Have all Project Status Reports for the time period been completed and submitted?
	4. Have all additions to, deletions from and modifications of the state data been made?
	5. Have all external responses been submitted?
	6. Have all necessary measures been gathered?
	7. Have all pertinent reviews for this process been completed?
	8. Have all action items generated during reviews that pertain to this process been completed?

Team members' signatures ______________ ______________

______________ ______________

Manager's signature ______________

Completion Condition Checklist:

Project: ____________________ **Process:** E10: Receive Stimuli

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Have all stimuli been moved to all of the correct state data?
	2. Have all other additions to, deletions from and modifications of the state data been completed?
	3. Have all necessary measures been gathered?
	4. Have all engineering team members who need the information received been notified?

Team members' signatures ________________ ________________

________________ ________________

Manager's signature ________________

Completion Condition Checklist:

Project: ____________________ **Process:** E11: Submit a Question or Issue

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Have all additions to, deletions from and modifications of the state data been made?
	2. Have all questions that cannot be resolved by the engineering team been formally submitted as responses?
	3. Have all necessary measures been made?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________________ **Process:** E12: Understand Problem Domain

Engineering Team Manager: ____________________ **Iteration #:** _______

Date Completed:	Completion Condition
	1. Has the project manager determined that a sufficient understanding of the problem domain has been gained to complete this process?
	2. Has the Specification team developed an approved copy of the specifications?
	3. Is there a clear definition of what is to be accomplished by the software?
	4. Has a clear description of all external interfaces for the software (converting all real-world stimuli into digital stimuli and digital responses into real-world responses) been determined?
	5. Is there a clear model which describes the problem domain?
	6. Has the model been judged as adequate to describe the problem domain?
	7. Do requirements express conceptual integrity (includes requirements generated during this activity as well as previously existing requirements)?
	8. Have all additions to, deletions from and modifications of the state data been made?
	9. Have all necessary measures been gathered?
	10. Have all external responses been submitted?
	11. Have all pertinent reviews for this process been completed (for example, if this process is following 2167A, the System Requirements Review and the System Design Review need to have been completed)?
	12. Have all action items generated during reviews that pertain to this process been completed?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________________ **Process:** E13: Understand Solution Domain

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Has the project manager determined that a sufficient understanding of the solution domain has been gained to complete this process?
	2. Has the Specification team developed an approved copy of the specifications?
	3. Is there a clear definition of what is to be accomplished by the software?
	4. Has a clear description of all external interfaces for the software (converting all real-world stimuli into digital stimuli and digital responses into real-world responses) been determined?
	5. Is there a clear model which describes the solution domain?
	6. Has the model been judged as adequate to describe the solution domain?
	7. Have all pertinent reviews for this process been completed (for example, if this process is following 2167A, the System Requirements Review and the System Design Review need to have been completed)?
	8. Have all additions to, deletions from and modifications of the state data been made?
	9. Have all necessary measures been gathered?
	10. Have all external responses been submitted?
	11. Have all action items generated during reviews that pertain to this process been completed?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________________ **Process:** E14: Write Specifications

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Are the Specifications complete, consistent and unambiguous to the extent possible?
	2. Have the Specifications been reviewed and accepted by the Certification and Development teams?
	3. Has the Specification team demonstrated that the Specifications describe a cost effective solution?
	4. Is the present version of the Specifications consistent with the Master Project Schedule, Project Charter, etc.?
	5. Have the Specifications been placed under configuration management?
	6. Are the Specifications consistent with the system requirements, and any software requirements that were submitted by management?
	7. Are the Specifications for the software consistent with the rest of the system, including hardware and human activities (for example, in terms of 2167A, these issues would need to be resolved by the System/Segment Design Document)?
	8. Are the Specifications consistent with the software development plan?
	9. Have the five volumes of the Specifications been placed in the correct state data?
	10. Have all necessary measures been gathered?
	11. Have all pertinent reviews for this process been completed?
	12. Have all action items generated during reviews that pertain to this process been completed?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________________ **Process:** E15: Write Construction Plan

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Will the Construction Plan meet the following constraints: a) resources, b) budget, c) schedule, d) provides the level of quality desired for the project?
	2. Has the Construction Plan decomposed the project into the best set of executable increments from the following standpoints: a) Size suitable to development teams' capabilities (all modules should normally be under 10 KSLOC barring exceptional circumstances), b) Complexity suitable to certification team's capabilities, c) Reuse, d) Prototyping, e) Complexity/size of the predicted solution domain, f) Hardware domain/operating environment of the system to be implemented, g) Client/deliverable obligations?
	3. Are the Specifications consistent with the software development plan?
	4. Has the Construction Plan been placed in the correct state data?
	5. Have all necessary measures been gathered?
	6. Have all pertinent reviews for this process been completed?
	7. Have all action items generated during reviews that pertain to this process been completed?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________________ **Process:** E16: Increment Development

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Have all necessary measures been gathered?
	2. Have all completion conditions for all four sub-processes (E18, E19, E20, E21) been achieved?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________________ **Process:** E17: Increment Certification

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Have all necessary measures been gathered?
	2. Have all completion conditions for all four sub-processes (E21, E22, E23, E24) been achieved?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________ **Process:** E18: Develop Increment i

Engineering Team Manager: ____________ **Iteration #:** ______

Date Completed:	Completion Condition
	Black, State and Clear boxes and code 1. Have team reviews been held for all boxes, trade studies and code?
	2. Have all boxes, trade studies and code passed their final review?
	3. Have all project/installation standards for design languages and code been adhered to, including: a) Using only the four basic structures b) Following language syntax conventions c) Providing function commentary where needed?
	4. Have all pertinent reviews for this process been completed?
	5. Have all action items generated during reviews that pertain to this process been completed?
	6. Are the items developed consistent with the software development plan?
	Black boxes 7. Have all black box functions been clearly defined in terms of stimulus histories, and have been defined using an acceptable format?
	8. Have all black box functions been verified to their specification (Note: A higher level clear box is a lower level black box's specification)?
	9. Does the black box function process all stimuli values, both valid and invalid?
	State boxes 10. Have all state box functions been validated to be equivalent to their black box functions?
	11. Does the state box function process all stimuli values, both valid and invalid leaving the state data in a valid state?

Team members' signatures ____________ ____________

____________ ____________

Manager's signature ____________

Completion Condition Checklist:

E18: Develop Increment i

Date Completed:	Completion Condition
	12. Have trade studies been conducted for all state data decisions: a) For elaboration at the selected level in the usage hierarchy? b) Does state data abstraction have right balance between execution speed, performance and verifiability?
	13. Has transaction closure been obtained for all state boxes?
	Clear boxes 14. Have all clear box functions been verified to be equivalent to their state box functions?
	15. Are all interfaces in the usage hierarchy well defined?
	16. Has transaction closure been obtained for all clear boxes?
	17. For concurrent clear boxes, has an appropriate resolve function been defined?
	18. Have all opportunities for common services been evaluated to determine proper modularization?
	Code 19. Have all procedures in the increment been verified/code read by more than one team member?
	20. Has code for all procedures been developed by stepwise refinement?
	21. Have all refinements been proven: a) Simple ones by direct assertion? b) Complicated ones by formal proof?
	Overall 22. Is the increment complete according to the items listed in the Construction Plan (Volume 6 of the Specifications)?
	23. Have all state data in the Software Development Files been correctly added, modified or deleted?
	24. Have all necessary measures been gathered?
	25. Have all responses been submitted?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________________ **Process:** E19: Develop Certification Tests for Increments 1..i

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Do test scenarios reflect the operational profile of the software to be tested?
	2. Can the results of executing all test scenarios be validated?
	3. Have test passage criteria for the increment (such as number of test scenarios, failures or reliability goals) been determined and corresponding test information generated?
	4. Have expected results been generated, and passage criteria determined for each test case?
	5. Have sufficient test scenarios been generated to certify the increment to the desired level of reliability, given the expected rate of failure?
	6. Is the increment complete according to the items listed in the Construction Plan (Volume 6 of the Specifications)?
	7. Has all state data in the Software Certification Files been correctly added, changed or deleted?
	8. Have all pertinent reviews for this process been completed?
	9. Have all action items generated during reviews that pertain to this process been completed?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________________ **Process:** E20: Update Specifications

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Does the present version of the Specifications represent the precise system the Development team is implementing and the Certification team is certifying?
	2. Is the present version of the Specifications consistent with the Master Project Schedule, Project Charter, etc.?
	3. Is the present version of the Specifications consistent with the software development plan?
	4. Are the Specifications for the software consistent with the rest of the system, including hardware and human activities (for example, in terms of 2167A, these issues would need to be resolved by the System/Segment Design Document)?
	5. Have all pertinent reviews for this process been completed?
	6. Have all action items generated during reviews that pertain to this process been completed?
	7. Have all additions, changes or deletions from the Software Specification Files been correctly done?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________________ **Process:** E21: Increase Understanding of Problem and Solution Domains as Required

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Has the project manager determined that a sufficient understanding of the problem and solution domains have been gained to continue with the project?
	2. Have solutions that effect the entire team been disseminated?
	3. Have all pertinent reviews for this process been completed?
	4. Have all action items generated during reviews that pertain to this process been completed?
	5. Have all information to be preserved been placed in the correct state data?
	6. Have all external responses been submitted?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________________ **Process:** E22: Build System with Increment j

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Have all components necessary to build increment j been received?
	2. Are all components under configuration control?
	3. Have all components received been compiled successfully?
	4. Have all con ponents received been assembled successfully with the existing system into an executable system?
	5. Have all faults found during compilation and linking been noted on failure report forms and returned to the Development team?
	6. Does every changed component have a corresponding Engineering Change Notice?
	7. ıs the process complete in regards to what is described in the Construction Plan (Volume 6 of the Specifications)?
	8. Have all information to be preserved been placed in the correct state data?
	9. Has neccesary metrics been gathered?
	10. Have all pertinent reviews for this process been completed (for example, if the project is following 2167A, the Function and Physical Configuration Audit needs to be completed)?
	11. Have all action items generated during reviews that pertain to this process been completed?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________________ **Process:** E23: Certify Increments 1..j

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Have all test executions been validated with the expected result for that test case?
	2. Has testing been stopped for one of the following reasons: a) All test scenarios executed, b) Observed failures to be corrected by Development team, c) Management decision?
	3. Have all failures found by the certification team led to a Failure Report being put in the Failure Report File, which will lead to Development team resolution of the Failure Report?
	4. Is the process complete in regards to what is described in the Construction Plan (Volume 6 of the Specifications)?
	5. Have all additions, deletions or modifications to the state data been completed?
	6. Have all necessary measures been gathered?
	7. Have all pertinent reviews for this process been completed?
	8. Have all action items generated during reviews that pertain to this process been completed?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________

Completion Condition Checklist:

Project: ____________________ **Process:** E24: Correct Code Increments 1..j

Engineering Team Manager: ____________________ **Iteration #:** ________

Date Completed:	Completion Condition
	1. Have all Software Failure Reports received by the Development team been resolved?
	2. Have engineering changes made to the code been carefully noted?
	3. Have all changes to the code been verified by Development team members, as well as by the Development team as a whole?
	4. Has every change made to the code also been propagated back throughout the clear, state and black box designs?
	5. Has an Engineering Change Notice been completed for every modification to the code?
	6. Have all pertinent reviews for this process been completed?
	7. Have all action items generated during reviews that pertain to this process been completed?

Team members' signatures ____________________ ____________________

____________________ ____________________

Manager's signature ____________________